From monoids to near-semirings

Tom Schrijvers

KU LEUVEN

The Essence of MonadPlus and Alternative

Tom Schrijvers

KU LEUVEN

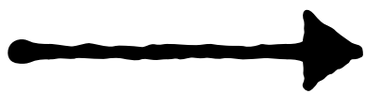
Joint work with

Exequiel Rivas

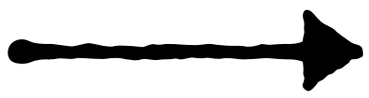
Mauro Jaskelioff

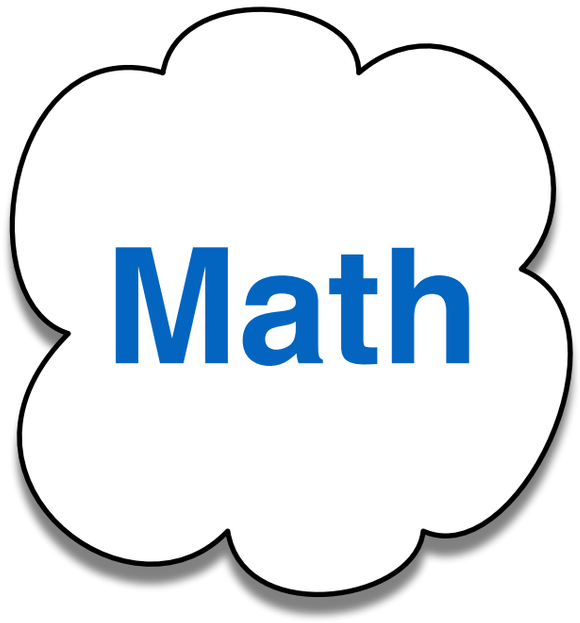
Overview

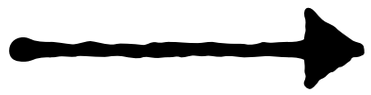
Abstract Patterns



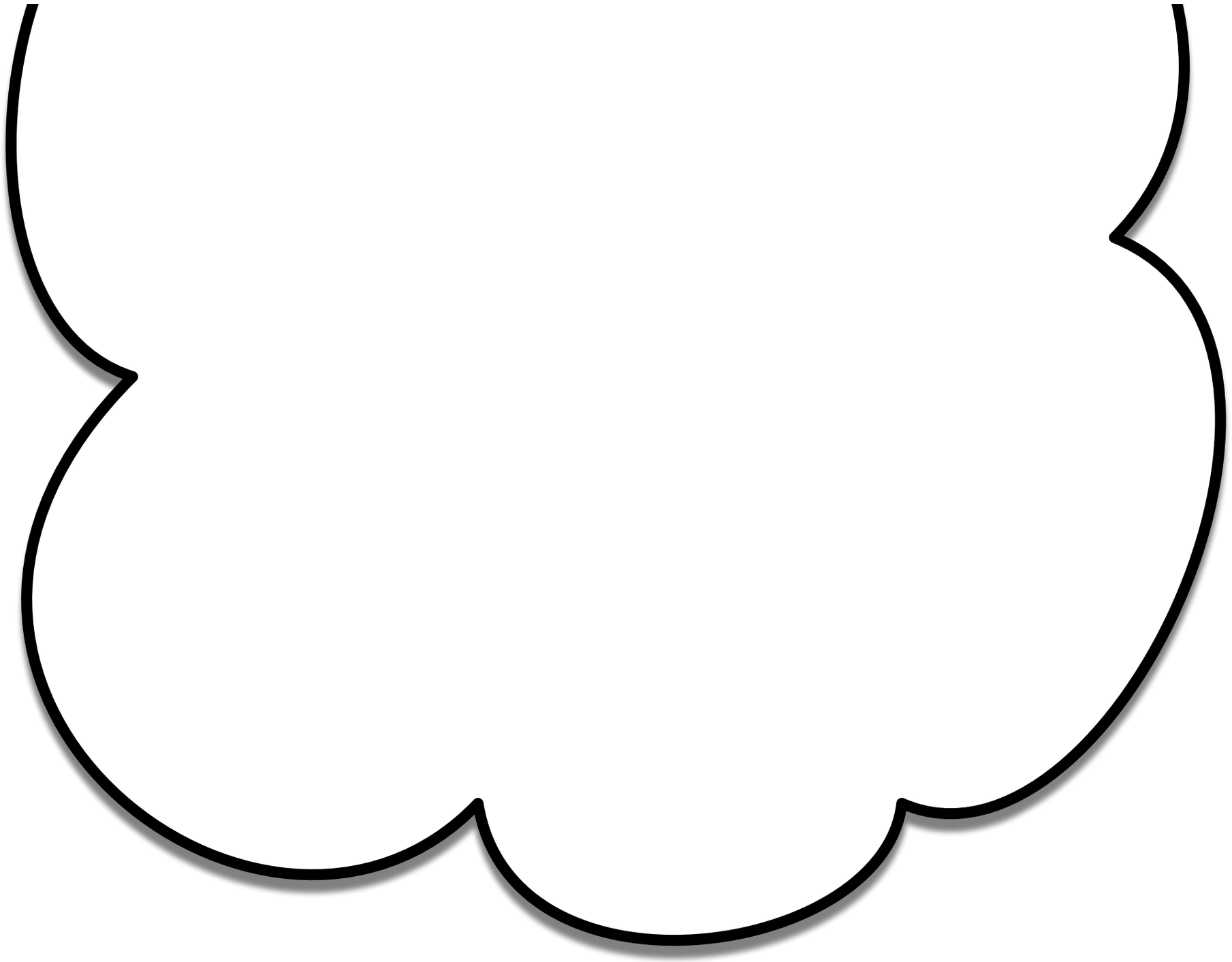
Abstract Patterns



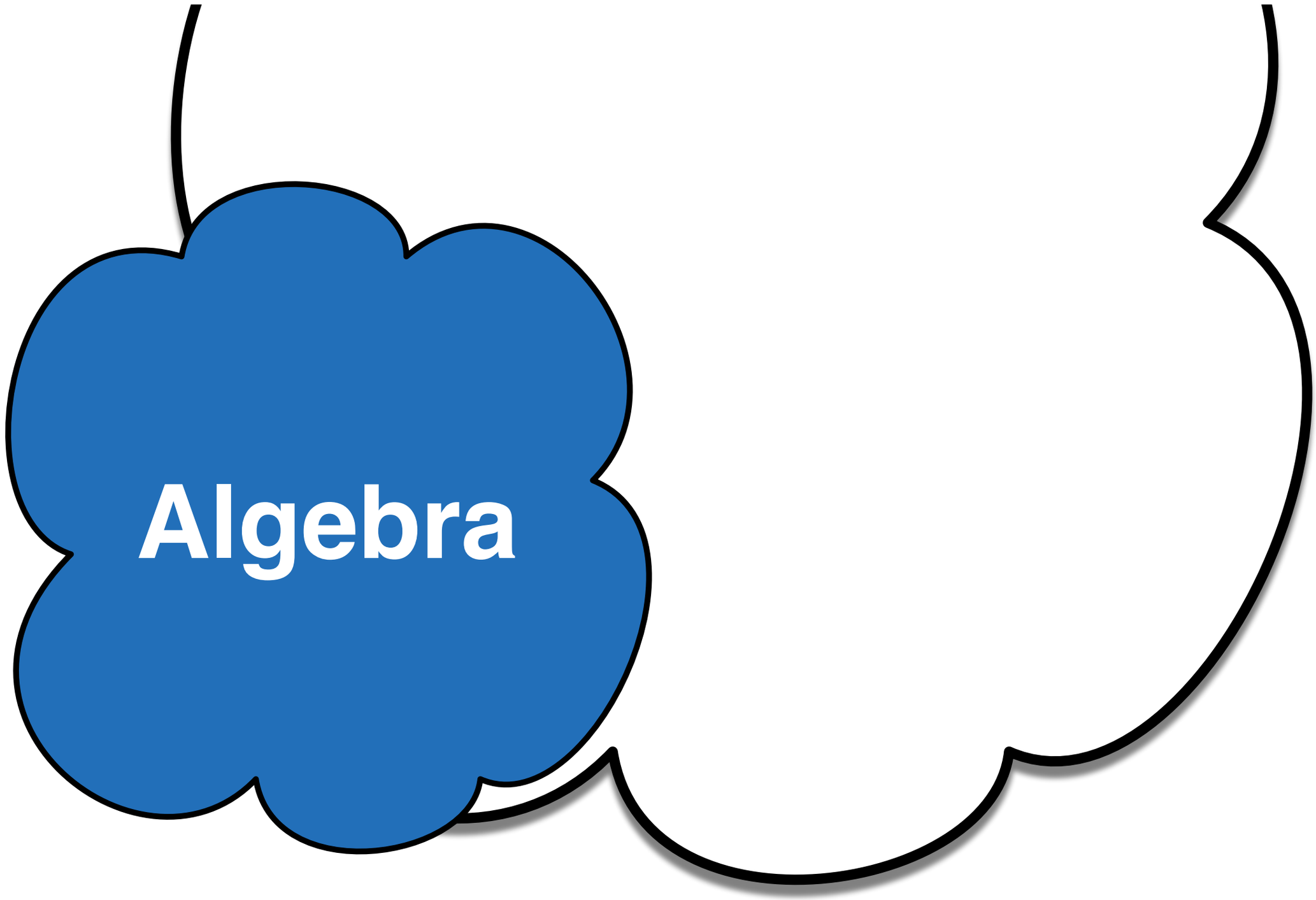




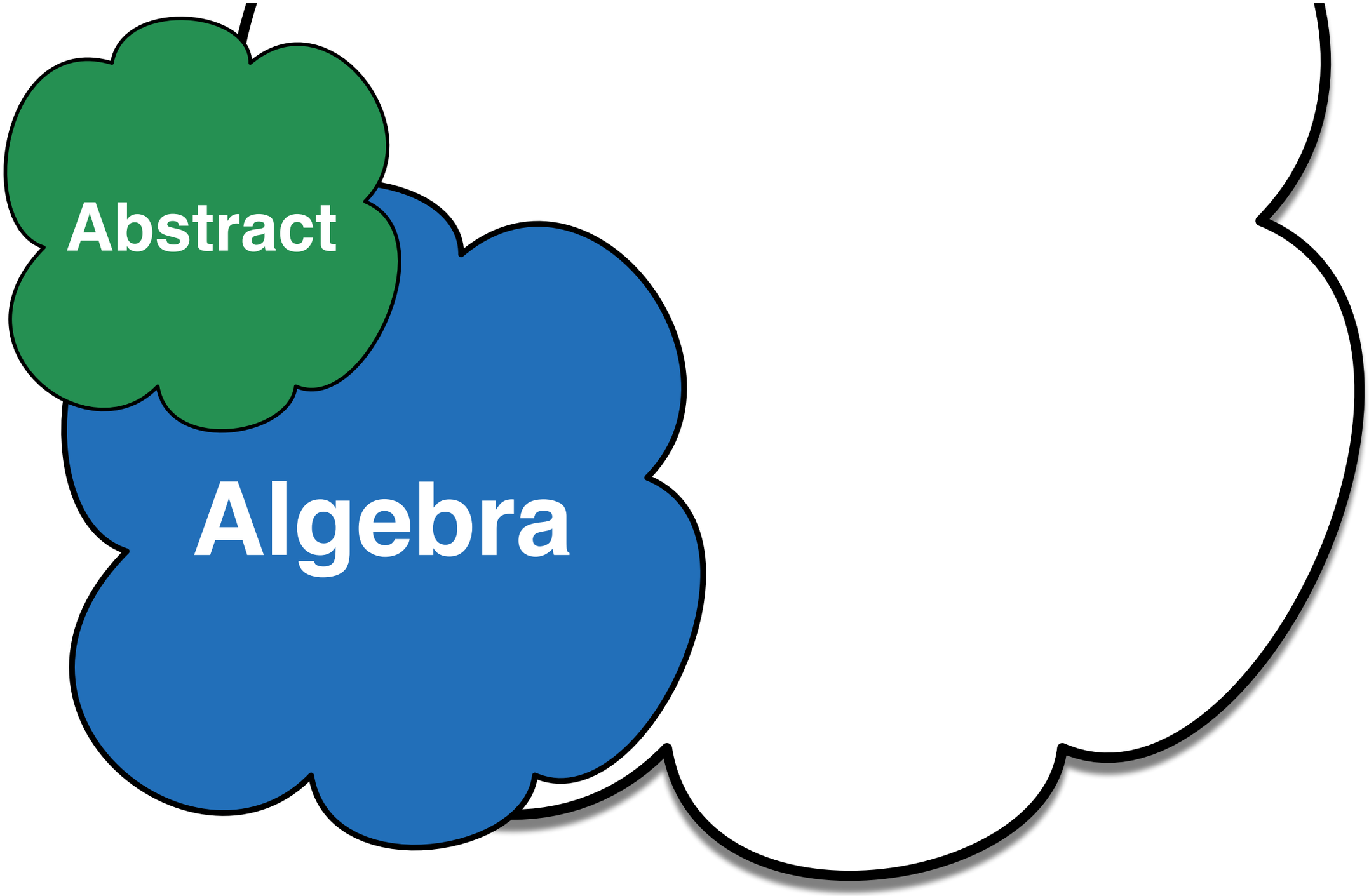
Haskell's Math Inspiration



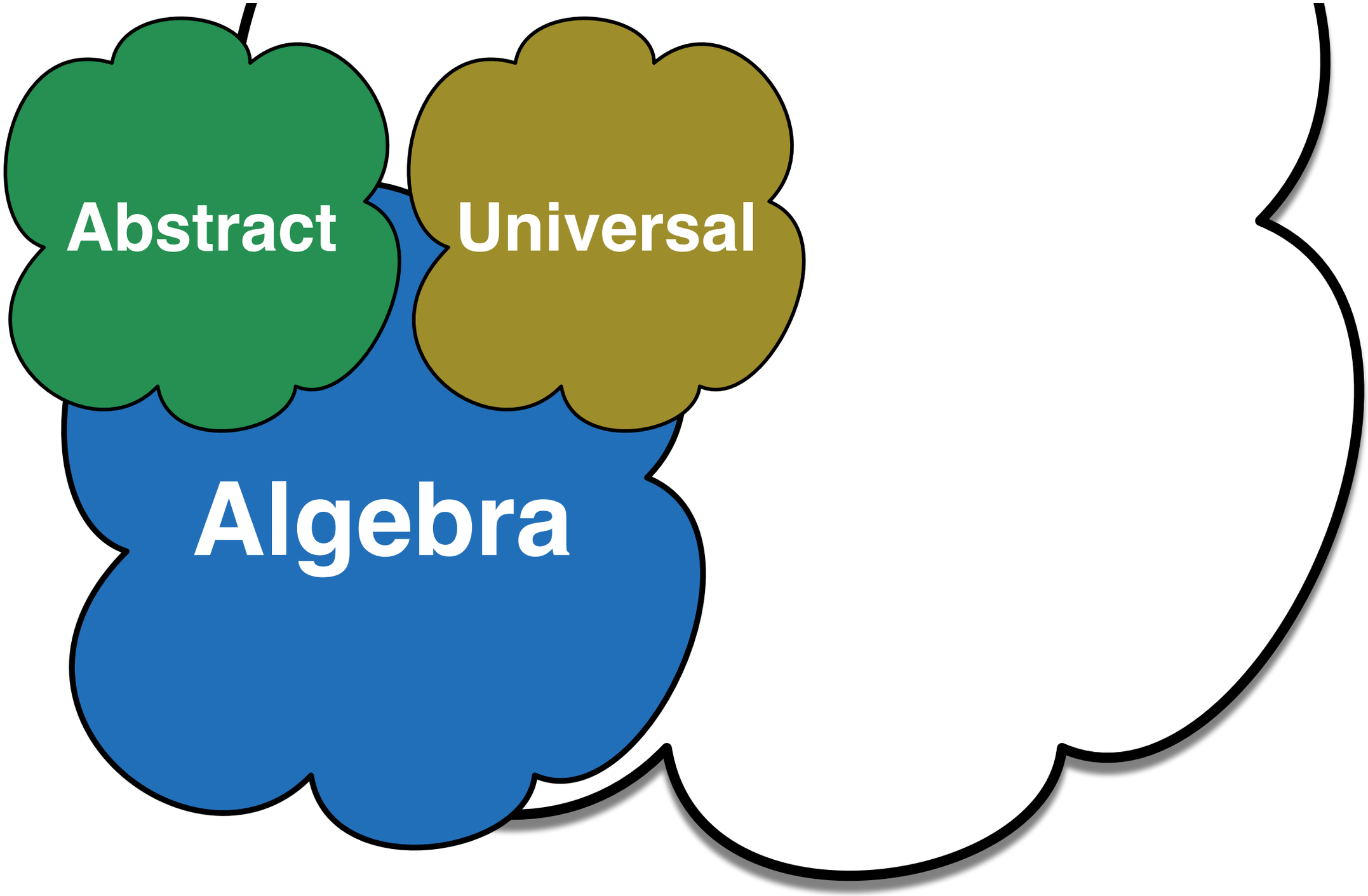
Haskell's Math Inspiration



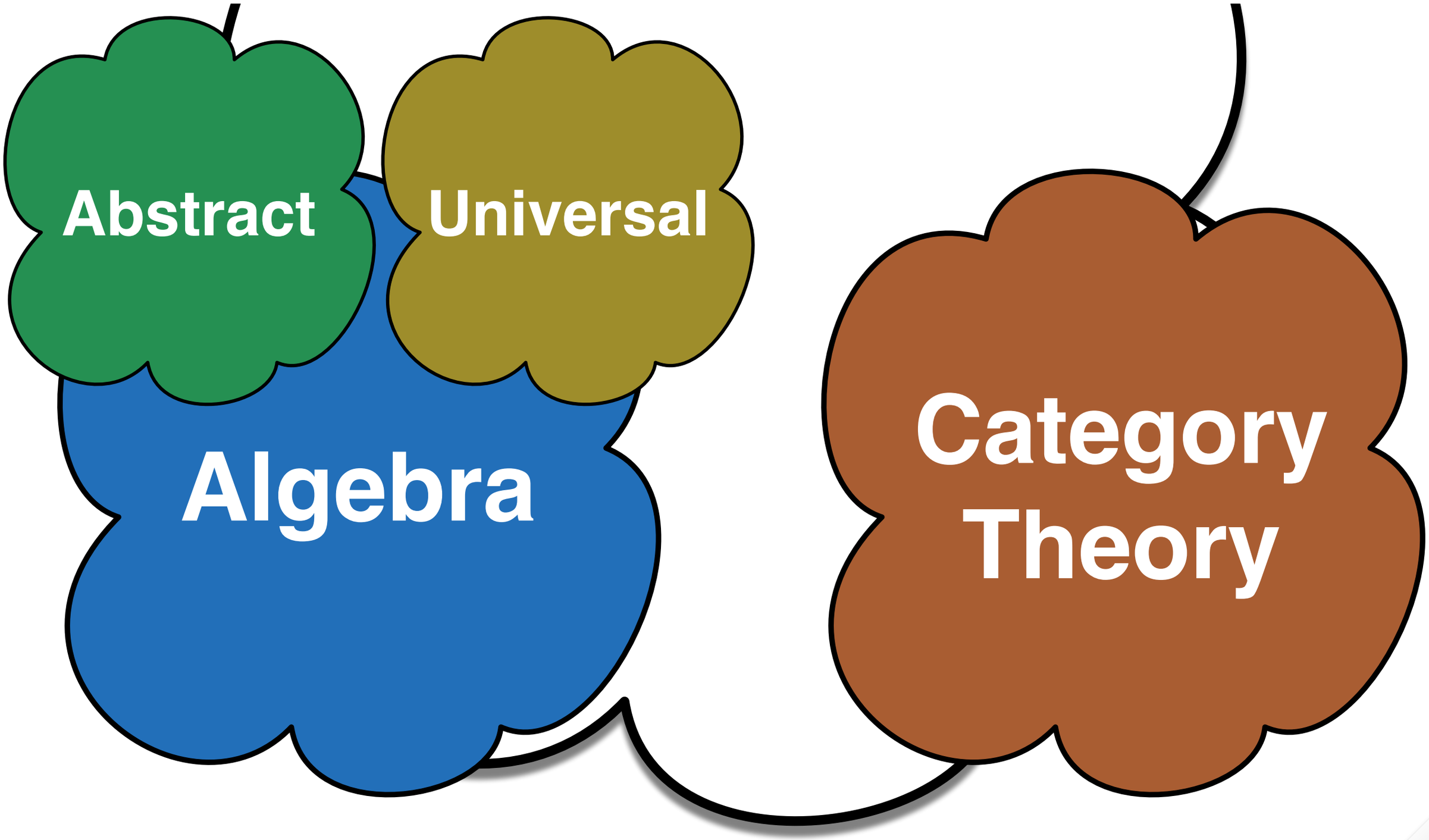
Haskell's Math Inspiration



Haskell's Math Inspiration



Haskell's Math Inspiration



Algebra

Category Theory

Algebra

Category Theory

regular
Haskell monoids

Algebra

Category Theory

regular Haskell monoids

monoids in the category of endofunctors

Algebra

Category Theory

regular
Haskell monoids

Haskell
monads and
applicative functors

Monoid

Monoid

generalise

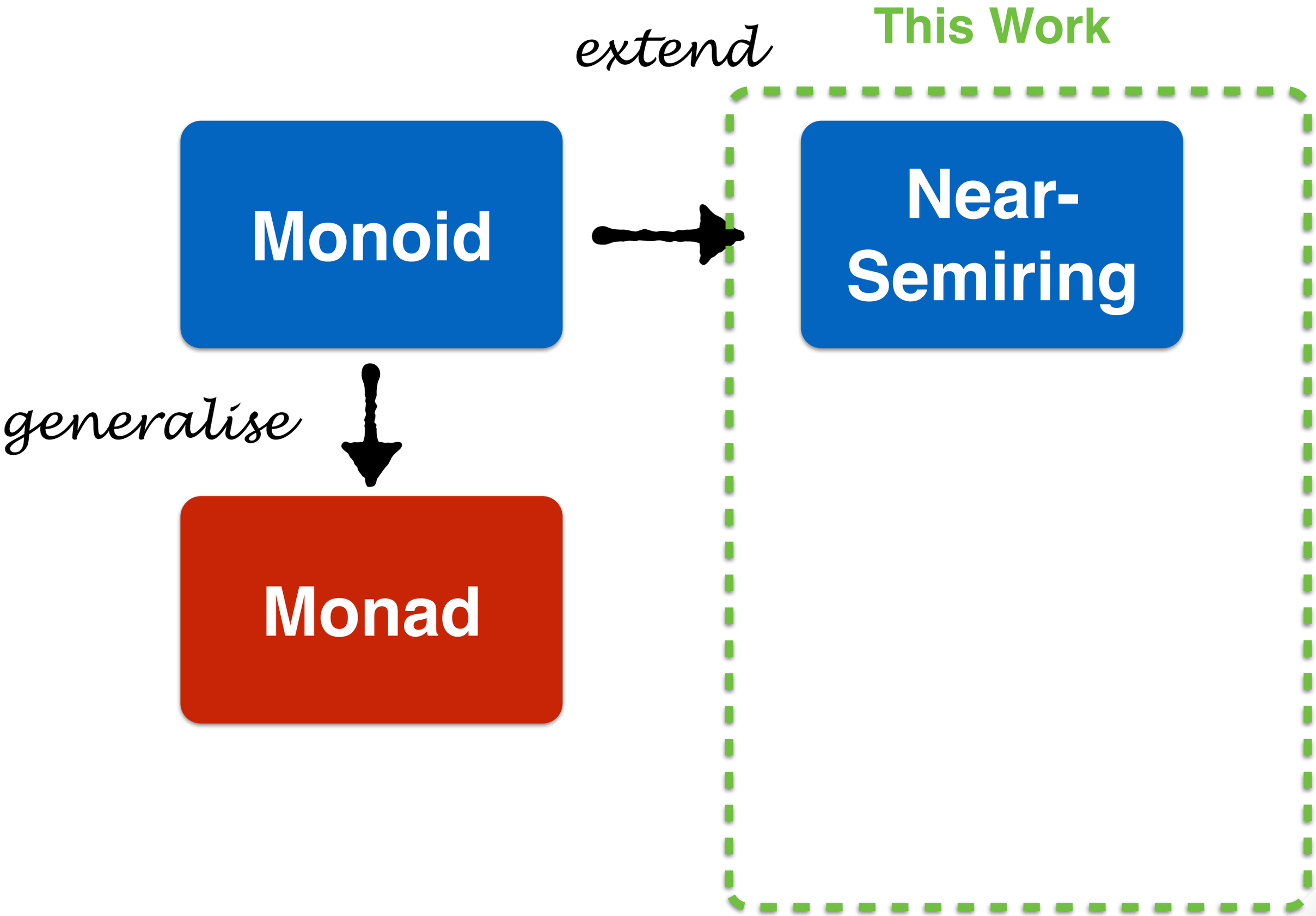
Monad

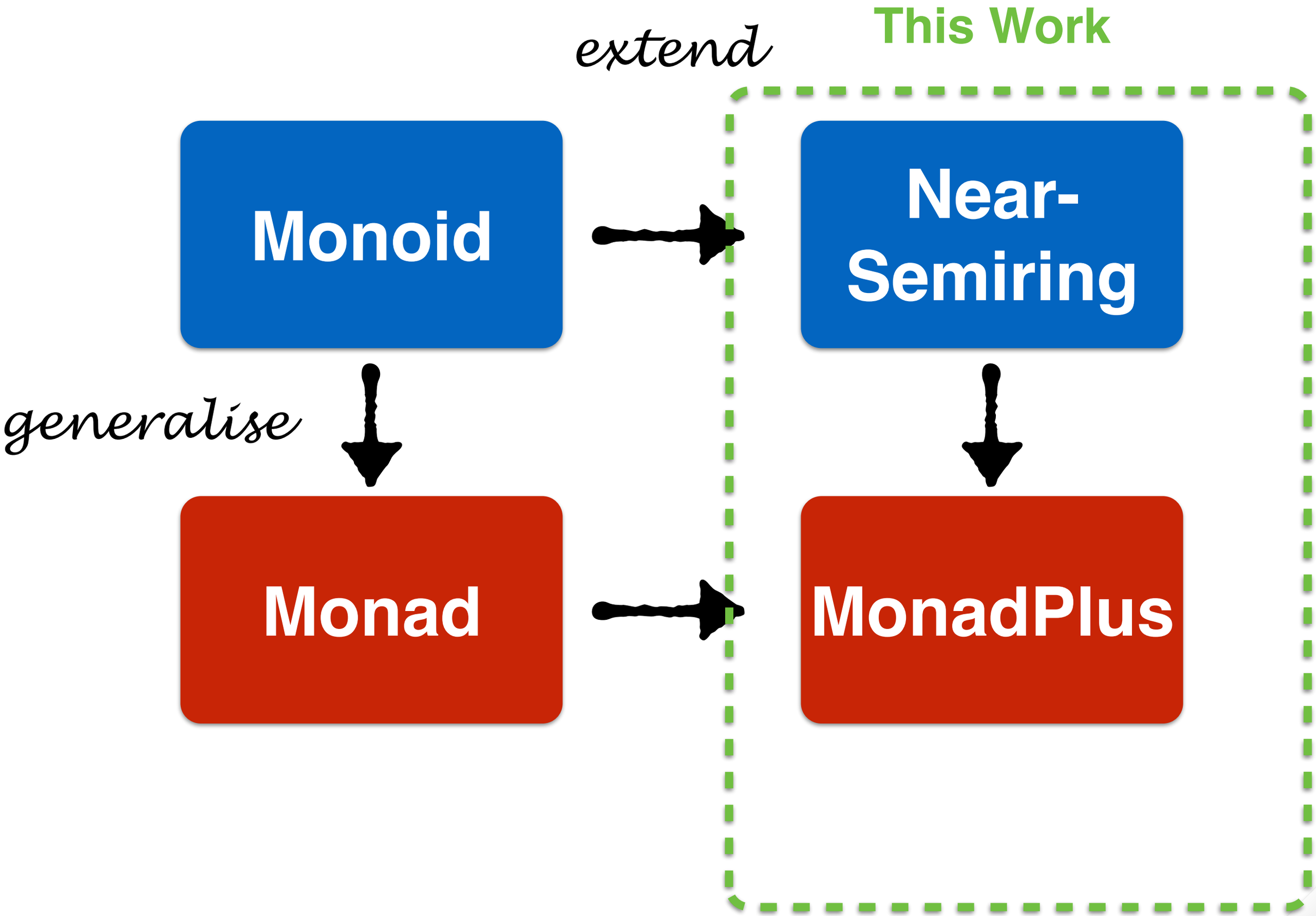
Monoid

generalise

Monad

This Work


This Work
extend
Monoid
Near-Semiring
generalise
Monad


This Work
extend
Monoid
Near-Semiring
generalise
Monad
MonadPlus

This Work

extend

Monoid

Near-Semiring

generalise

Monad

Applicative Functor

MonadPlus

Alternative

Monoids

The Monoid Structure
$(M, \otimes, \varepsilon)$

4
ingredients

The Monoid Structure $(M, \otimes, \varepsilon)$

A set
M

4
ingredients

The Monoid Structure $(M, \otimes, \varepsilon)$

A set
M

A binary operator
$\otimes$: M × M → M

4
ingredients

The Monoid Structure
$(M, \otimes, \varepsilon)$

A set
M

A binary operator
$\otimes$: M × M → M

4
ingredients

An element
$\varepsilon \in$ M

The Monoid Structure $(M, \otimes, \varepsilon)$

A set
M

A binary operator
$\otimes$: M × M → M

4 ingredients

An element
$\varepsilon \in$ M

3 Properties

The Monoid Structure $(M, \otimes, \varepsilon)$

A set
M

A binary operator
$\otimes$: M × M → M

$\mathbb{Z} = \{\ldots,-1,0,1,\ldots\}$

4 ingredients

An element
$\varepsilon \in$ M

3 Properties

The Monoid Structure $(M, \otimes, \varepsilon)$

A set
M

A binary operator
$\otimes$: M × M → M

$\mathbb{Z} = \{\ldots,-1,0,1,\ldots\}$

$+ : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$

4 ingredients

An element
$\varepsilon \in$ M

3 Properties

The Monoid Structure $(M, \otimes, \varepsilon)$

A set
M

A binary operator
$\otimes$: M × M → M

$\mathbb{Z} = \{\ldots,-1,0,1,\ldots\}$

$+ : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$

4 ingredients

$0 \in \mathbb{Z}$

An element
$\varepsilon \in$ M

3 Properties

The Monoid Structure $(M, \otimes, \varepsilon)$

A set
M

A binary operator
$\otimes$: M × M → M

$\mathbb{Z} = \{\ldots,-1,0,1,\ldots\}$

$+ : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$

4 ingredients

Left Unit

$\varepsilon \otimes x = x$

$0 \in \mathbb{Z}$

An element
$\varepsilon \in M$

3 Properties

The Monoid Structure $(M, \otimes, \varepsilon)$

A set
M

$\mathbb{Z} = \{\ldots,-1,0,1,\ldots\}$

A binary operator
$\otimes$: M × M → M

$+ : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$

4 ingredients

Left Unit **$\varepsilon \otimes x = x$**

Right Unit **$x \otimes \varepsilon = x$**

$0 \in \mathbb{Z}$

An element
$\varepsilon \in M$

3 Properties

The Monoid Structure $(M, \otimes, \varepsilon)$

A set
M

$\mathbb{Z} = \{\ldots,-1,0,1,\ldots\}$

An element
$\varepsilon \in M$

$0 \in \mathbb{Z}$

A binary operator
$\otimes : M \times M \rightarrow M$

$+ : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$

4 ingredients

3 Properties

Left Unit **$\varepsilon \otimes x = x$**

Right Unit **$x \otimes \varepsilon = x$**

Associativity **$x \otimes (y \otimes z) = (x \otimes y) \otimes z$**

Monoid Constructions

Monoid Constructions

Free Monoid

Monoid (Homo)morphism

a function between monoids

```
f :: M1 -> M2
```

such that:

```
f ε     = ε
```

and:

```
f (x ⊗ y) = f x ⊗ f y
```

Monoid (Homo)morphism

a function between monoids

```
length :: [a] -> Int
```

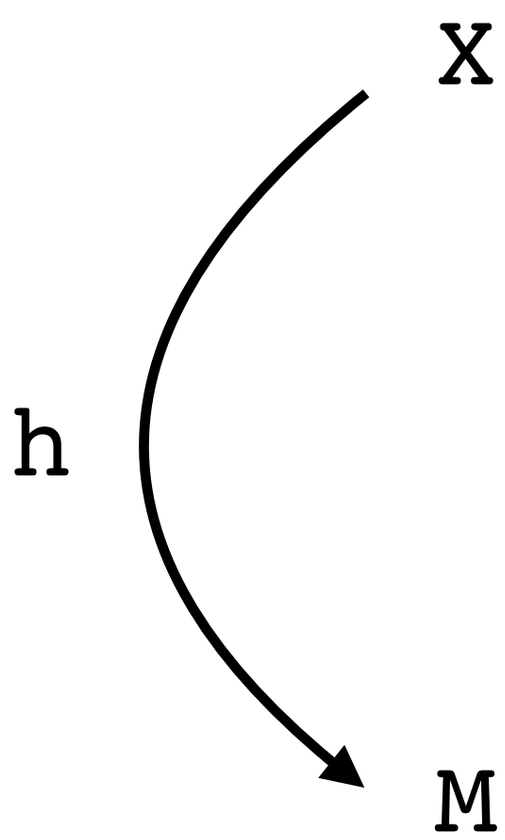
such that:

```
length [] = 0
```

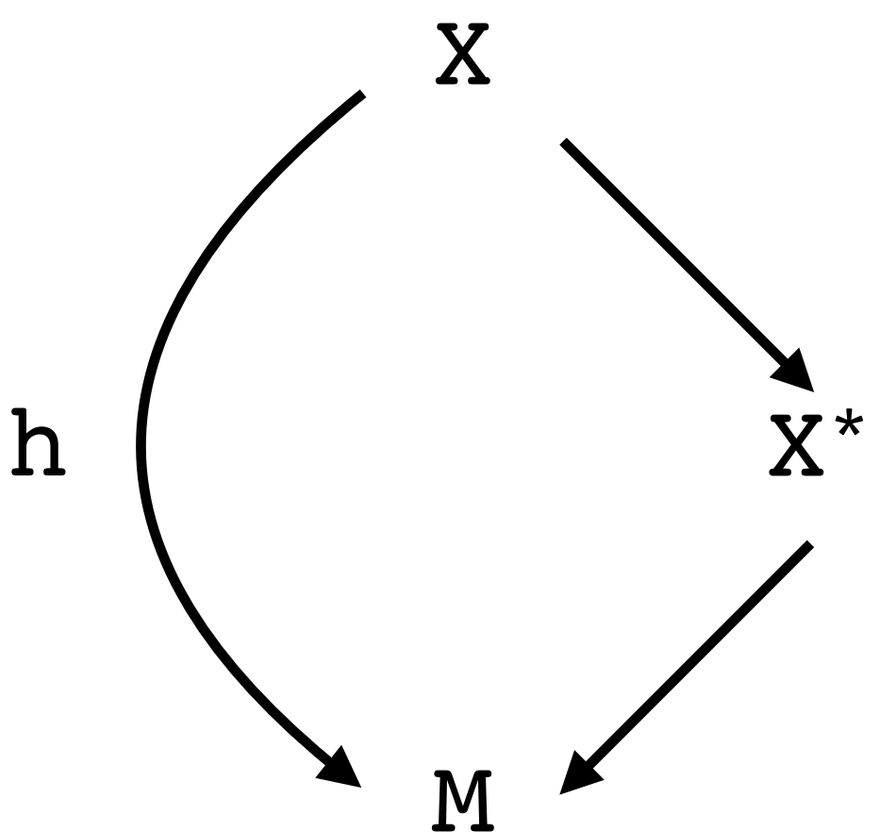
and:

```
length (x ++ y) = length x + length y
```

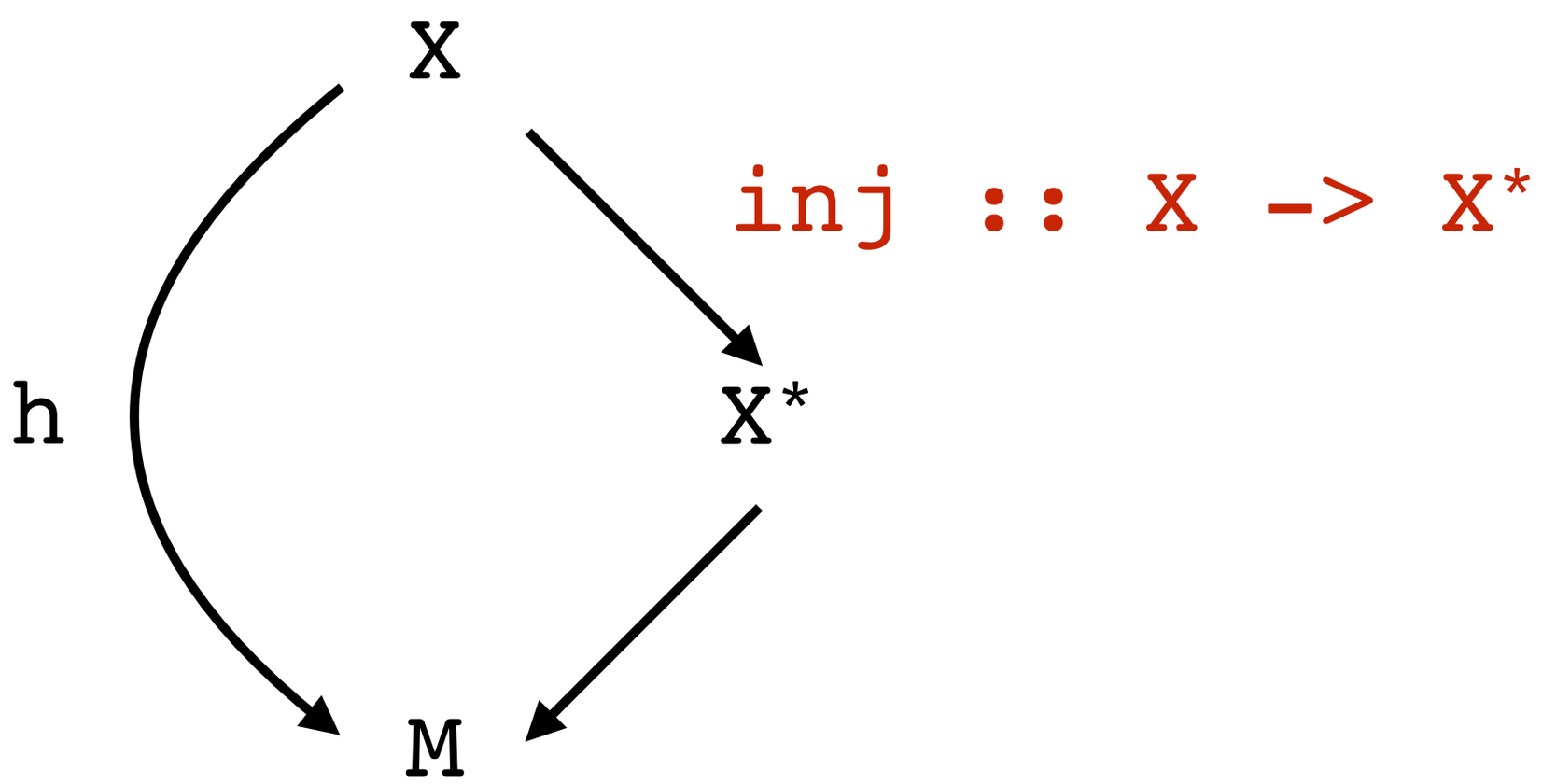
Free Monoid
$(X^*, \otimes, \varepsilon)$



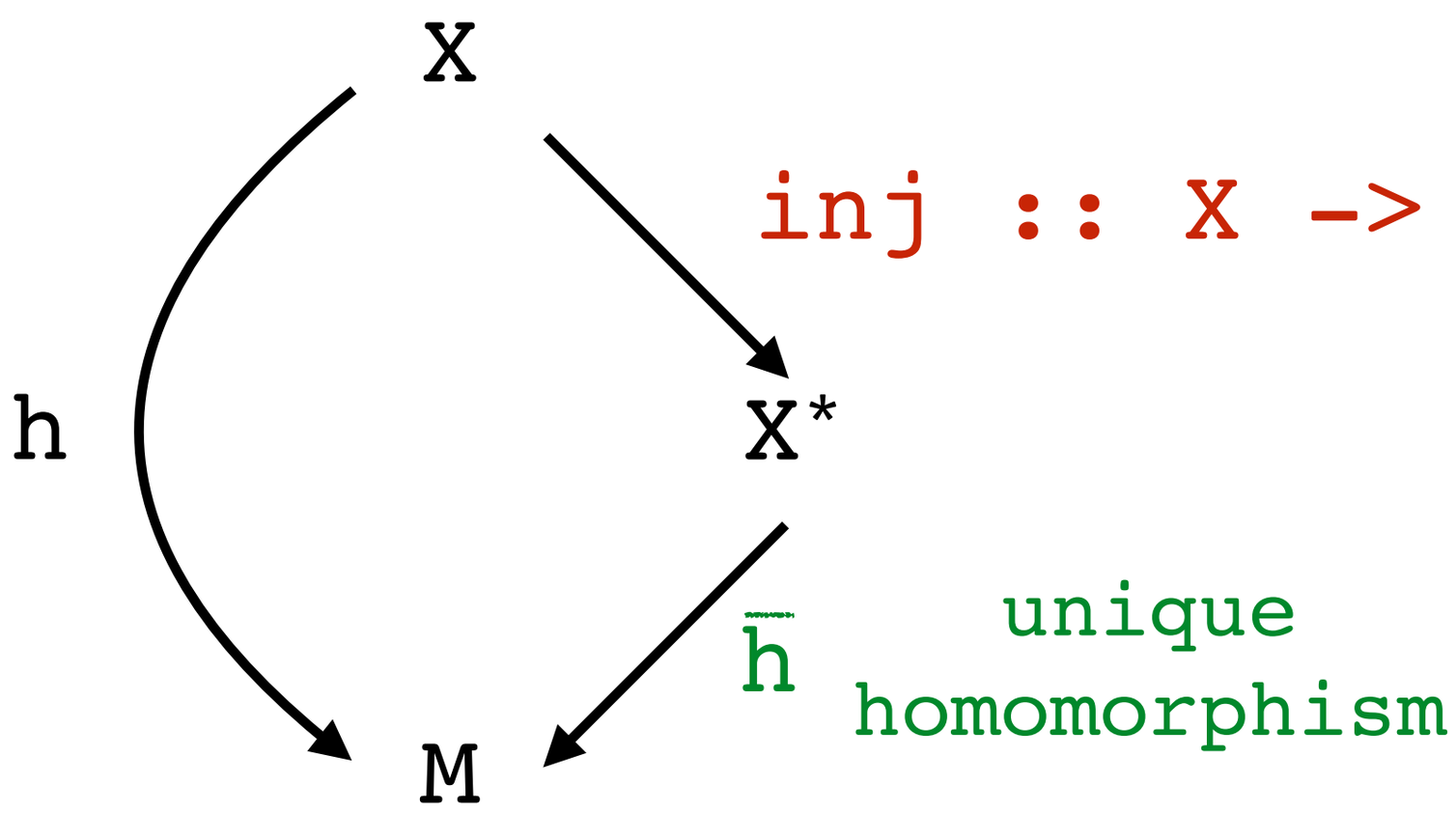
Free Monoid
$(X^*, \otimes, \varepsilon)$



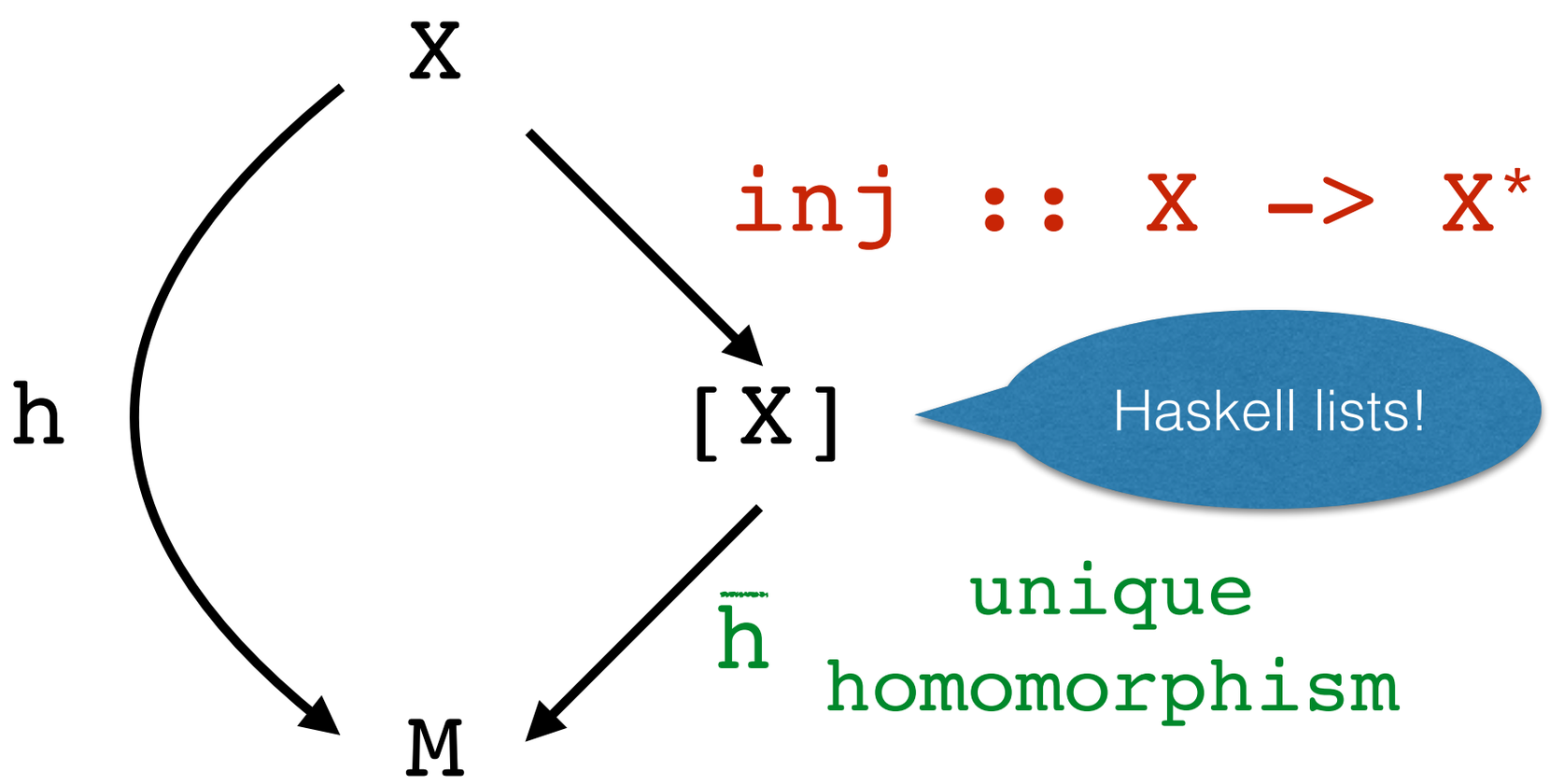
Free Monoid
$(X^*, \otimes, \varepsilon)$



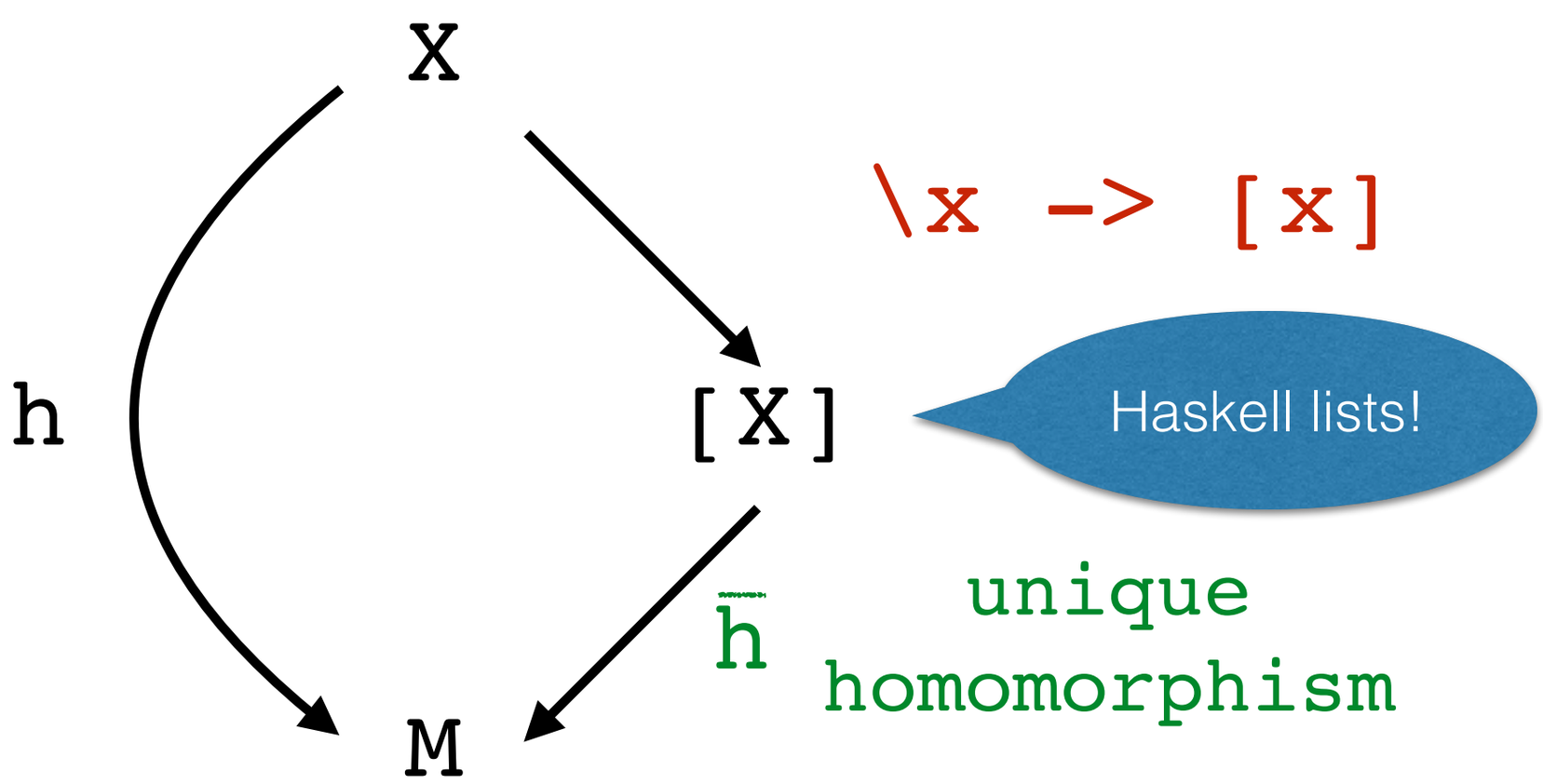
Free Monoid
$(X^*, \otimes, \varepsilon)$



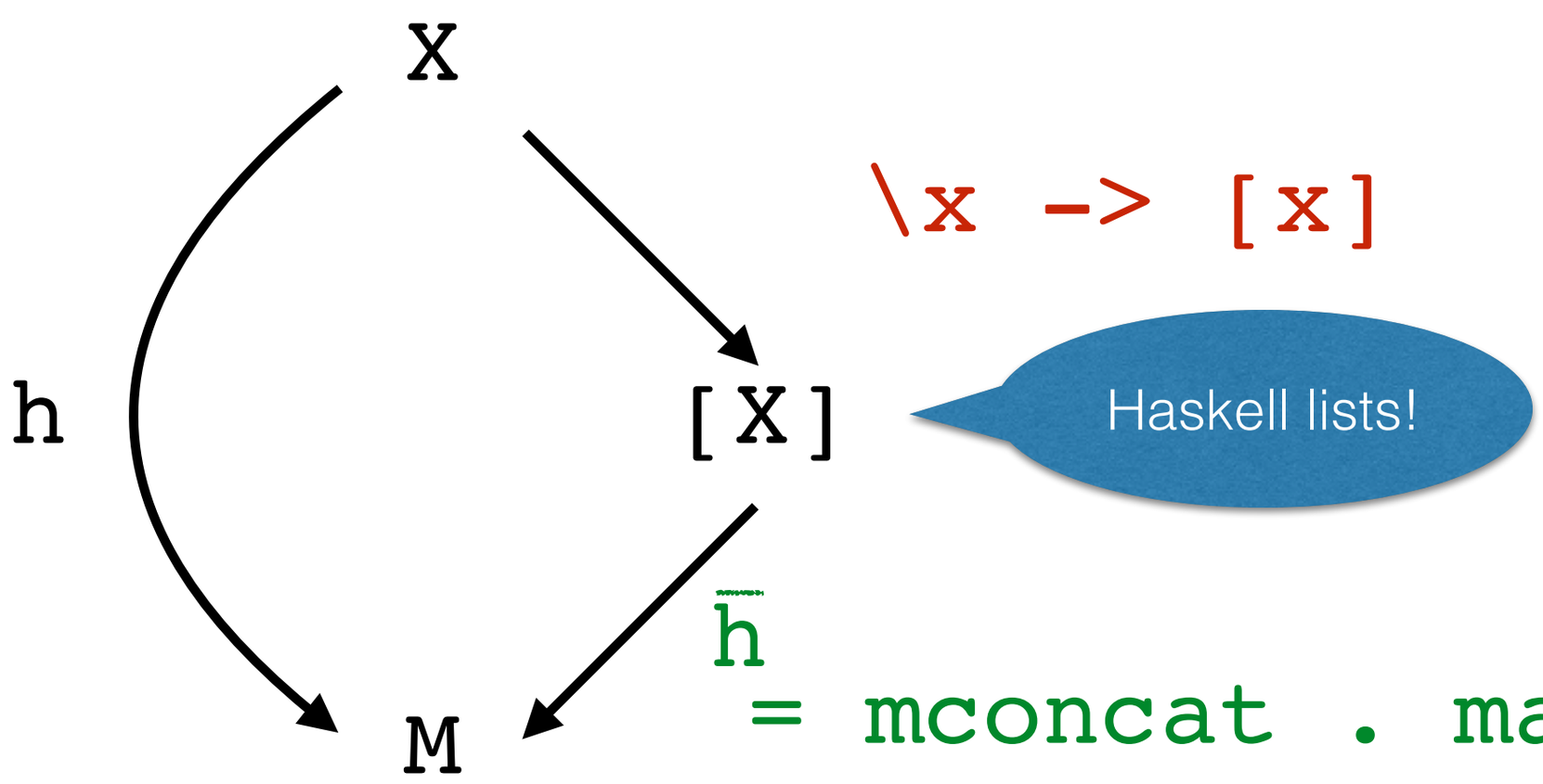
Free Monoid
$(X^*, \otimes, \varepsilon)$



Free Monoid
$(X^*, \otimes, \varepsilon)$

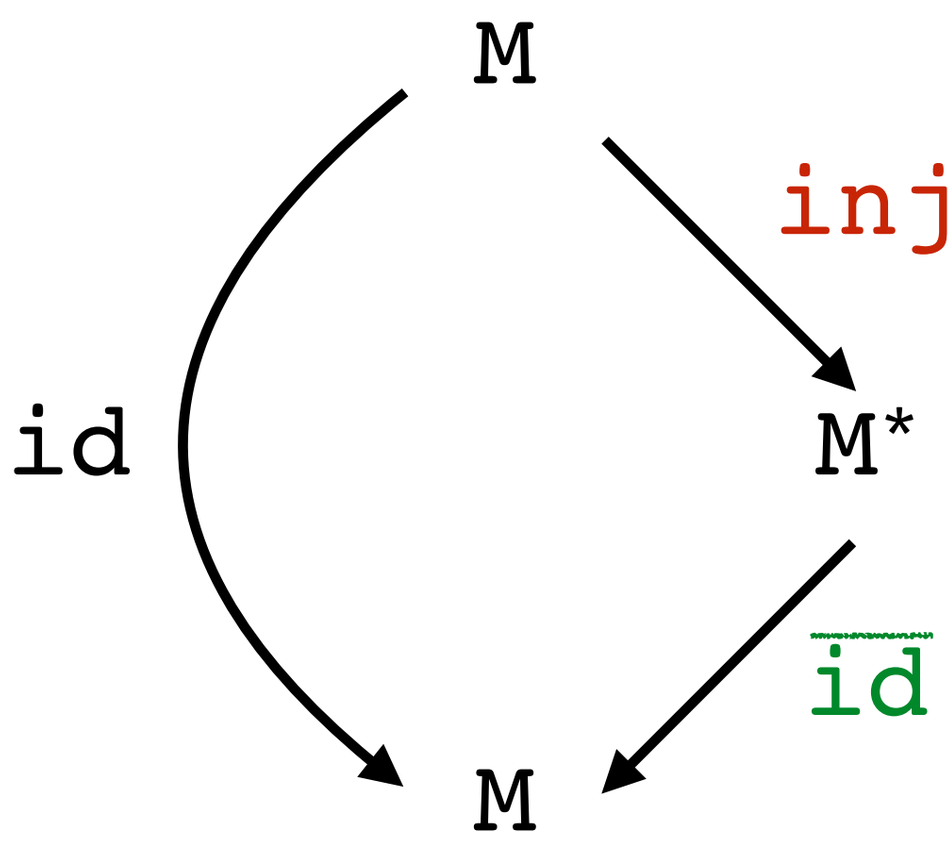


Free Monoid
$(X^*, \otimes, \varepsilon)$



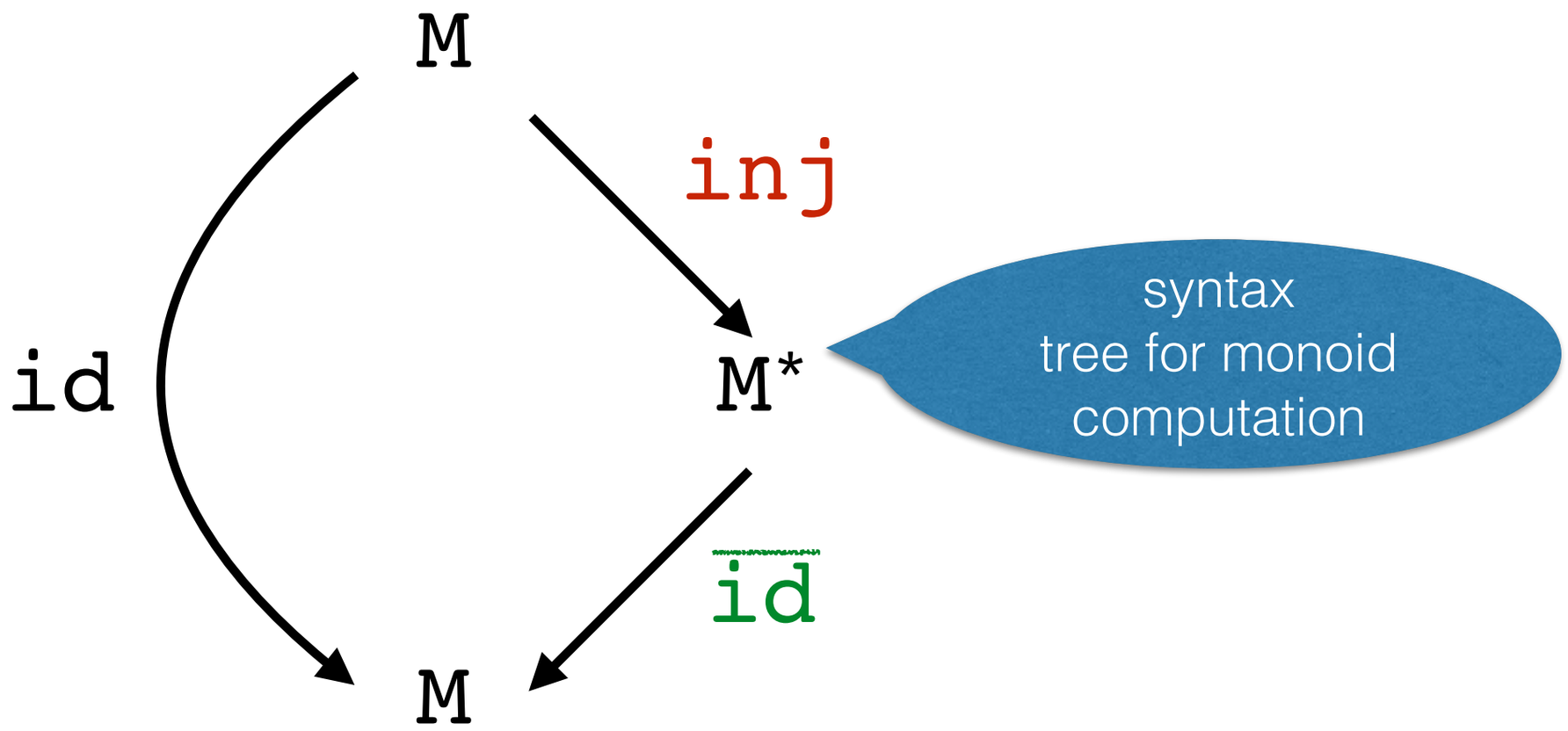
Free Monoid
$(M^*, \otimes, \varepsilon)$

X = M



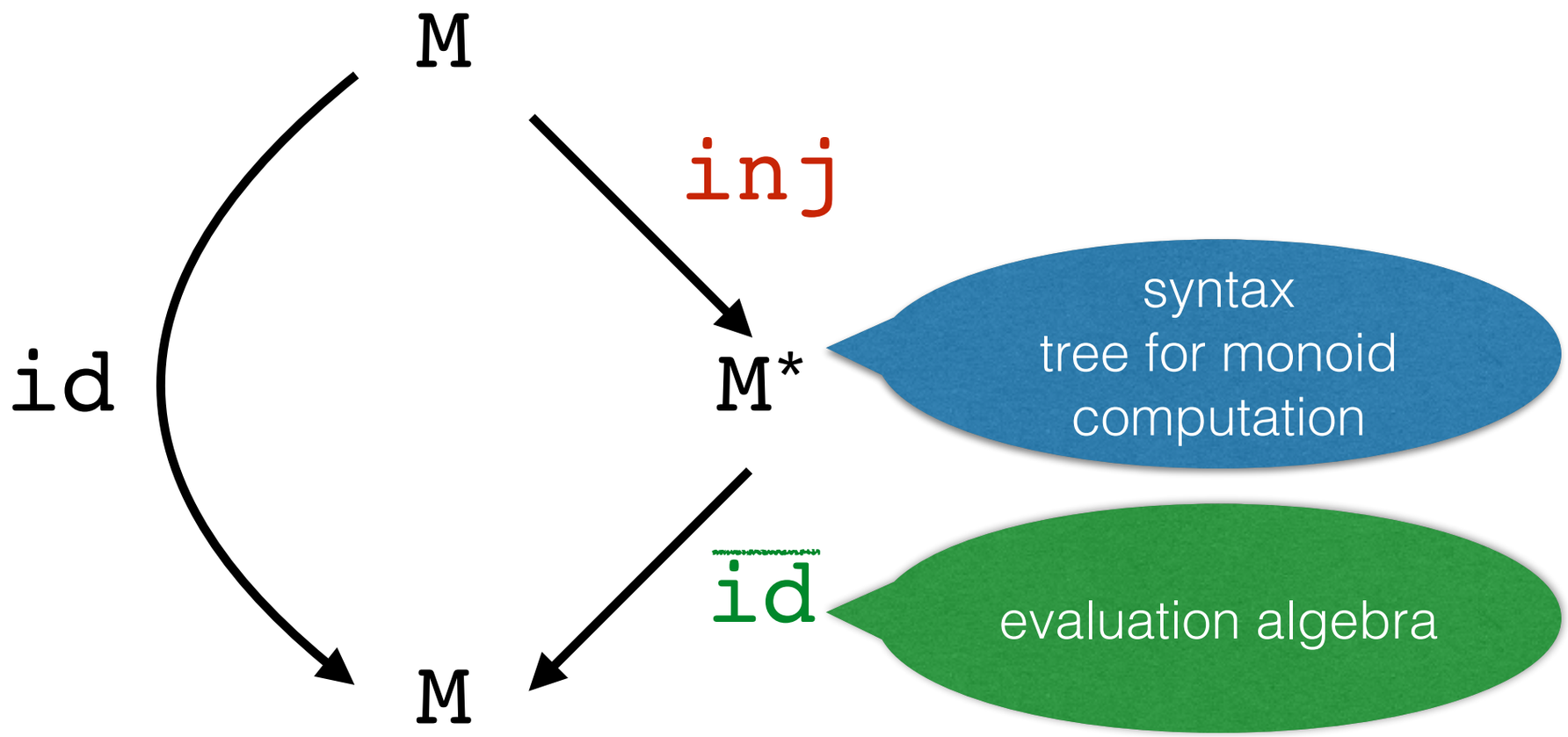
Free Monoid
$(M^*, \otimes, \varepsilon)$

X = M



Free Monoid
$(M^*, \otimes, \varepsilon)$

X = M



Monoid Constructions

Free Monoid

Monoid Constructions

Free Monoid

Cayley Representation

Equal, but not the same

Left Nested

$$(l_1 \mathbin{+\!\!+} l_2) \mathbin{+\!\!+} l_3$$

$$=$$

Right Nested

$$l_1 \mathbin{+\!\!+} (l_2 \mathbin{+\!\!+} l_3)$$

Equal, but not the same

slow

Left Nested $(l_1 \mathbin{+\!\!+} l_2) \mathbin{+\!\!+} l_3$

$=$

Right Nested $l_1 \mathbin{+\!\!+} (l_2 \mathbin{+\!\!+} l_3)$

fast

Equal, but not the same

slow

$2 * |l_1| + |l_2|$ steps

Left Nested $(l_1 ++ l_2) ++ l_3$

$=$

Right Nested $l_1 ++ (l_2 ++ l_3)$

fast

Equal, but not the same

slow

$2 * |l_1| + |l_2|$ steps

Left Nested $(l_1 ++ l_2) ++ l_3$

$=$

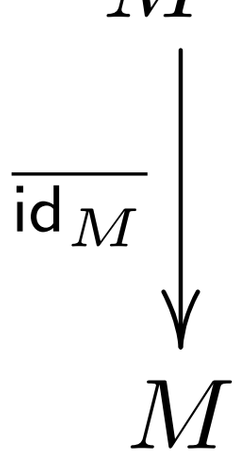
Right Nested $l_1 ++ (l_2 ++ l_3)$

fast

$|l_1| + |l_2|$ steps

A Better Representation?

Original Program



A Better Representation?

Original Program

M^*

slow evaluation

$\overline{\mathsf{id}_M}$

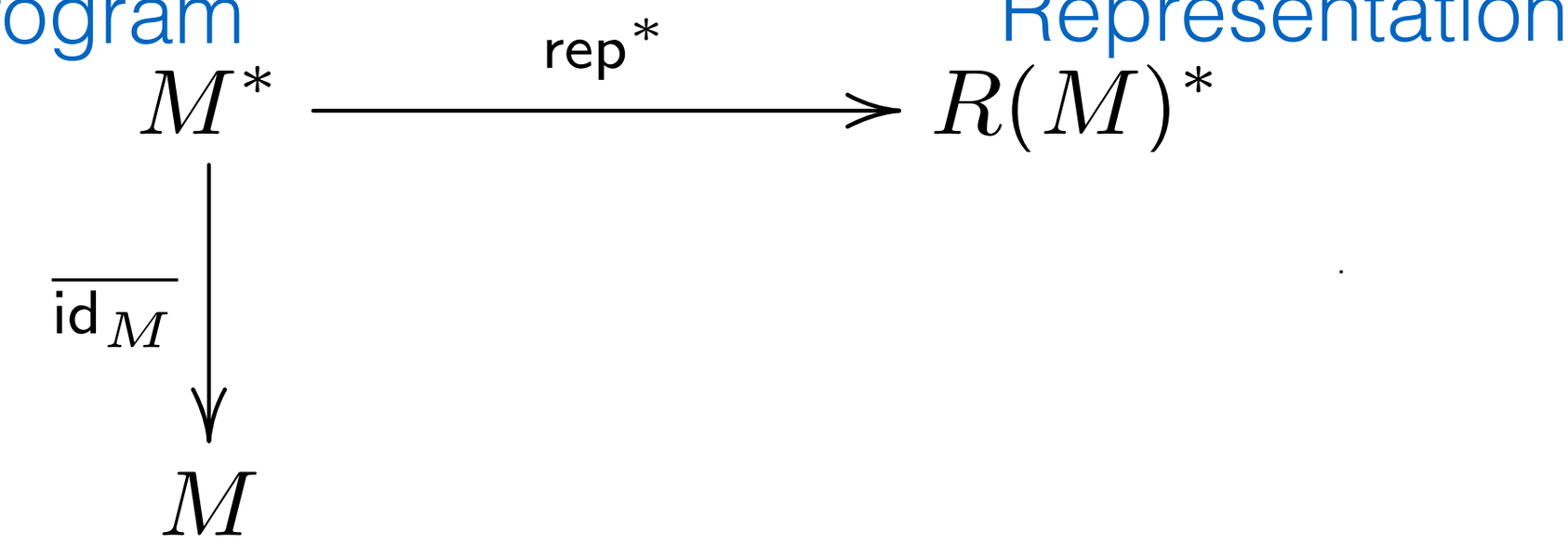
M

A Better Representation?

change
of representation

Original
Program

Better
Representation



A Better Representation?

change
of representation

Original
Program

Better
Representation

$$M^* \xrightarrow{\mathsf{rep}^*} R(M)^*$$

slow
evaluation

$$\downarrow \overline{\mathsf{id}_M}$$

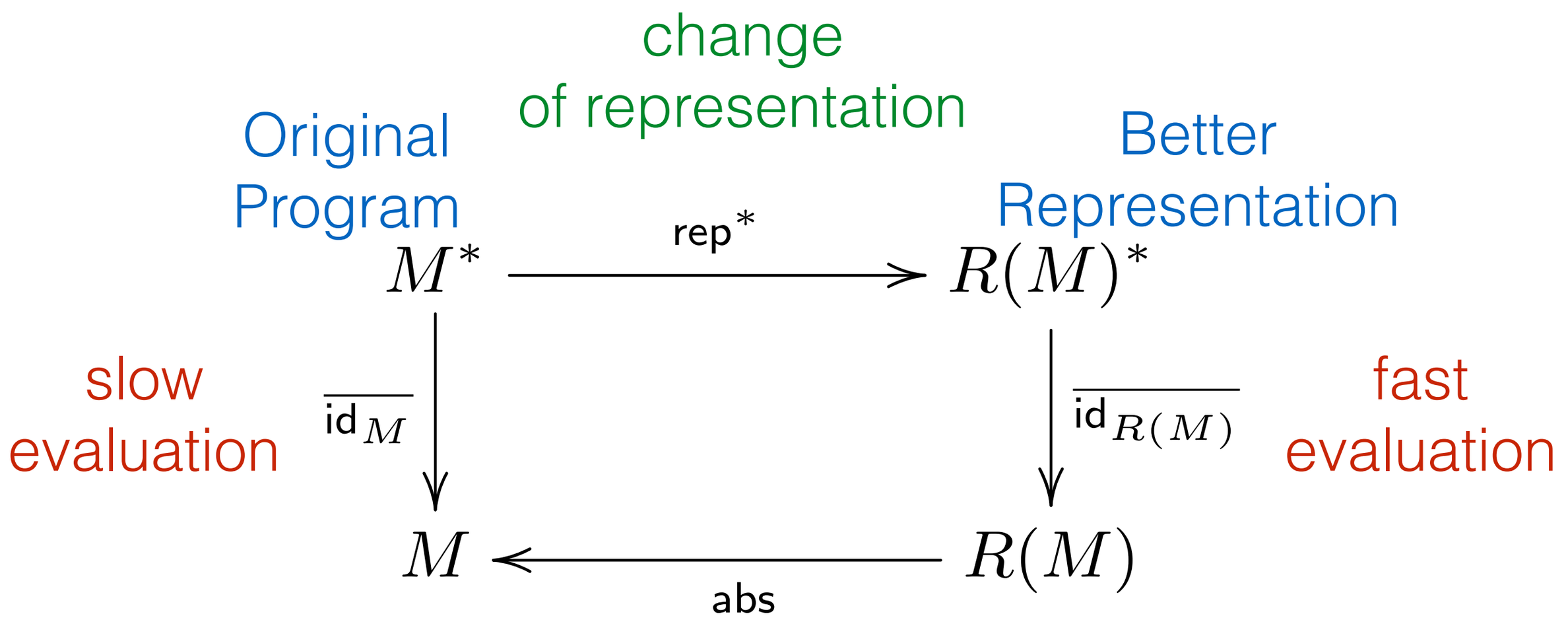
$$M$$

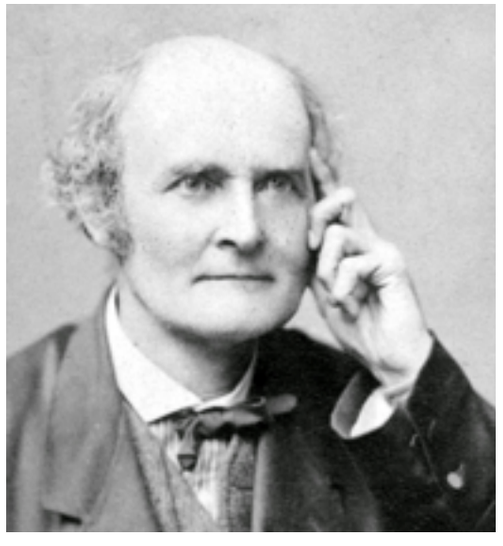
$$\downarrow \overline{\mathsf{id}_{R(M)}}$$

$$(M)$$

fast
evaluation

A Better Representation?

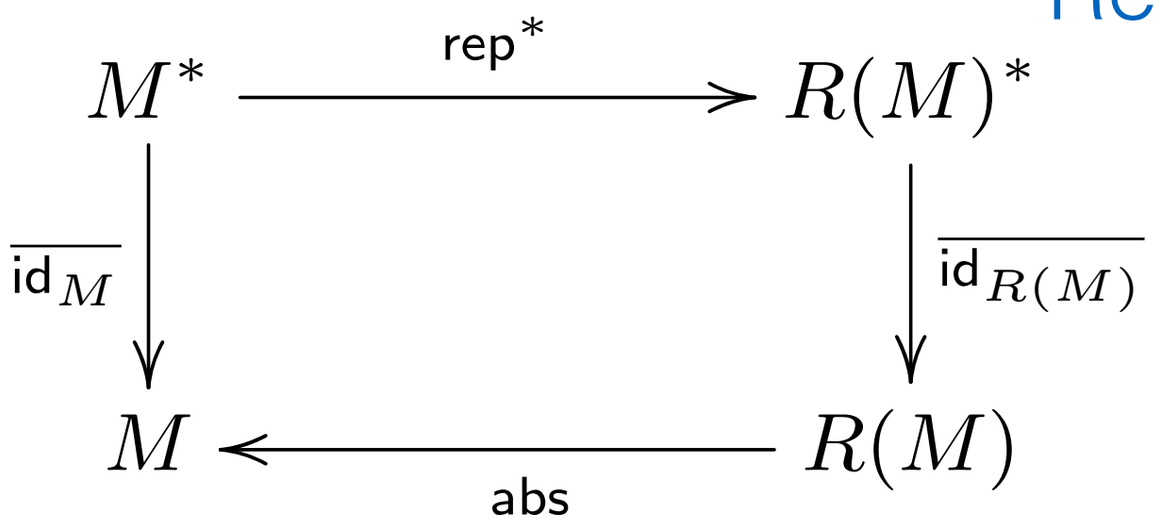


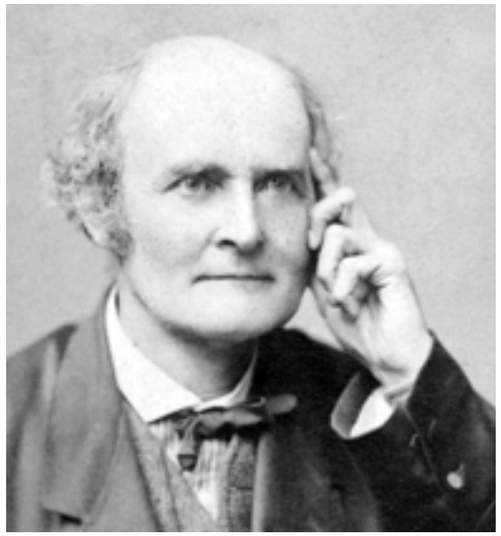


Cayley Representation

Original Program

Better Representation





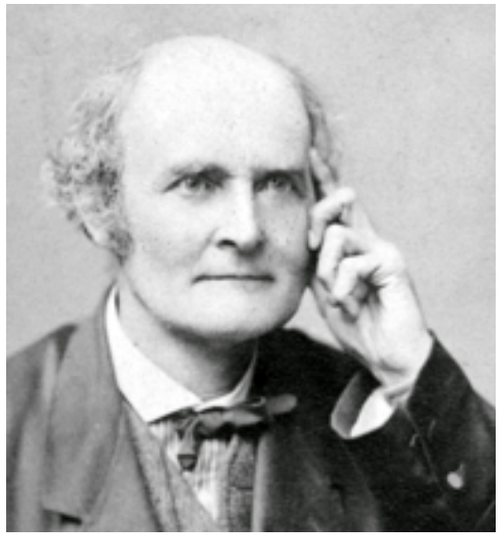
Cayley Representation

Original Program

```
rep(x) = λy.x ⊗ y
```

Better Representation

$$\begin{array}{ccc} M^* & \xrightarrow{\text{rep}^*} & R(M)^* \\ \Big\downarrow{\scriptstyle \overline{\text{id}_M}} & & \Big\downarrow{\scriptstyle \overline{\text{id}_{R(M)}}} \\ M & \xleftarrow[\text{abs}]{} & R(M) \end{array}$$



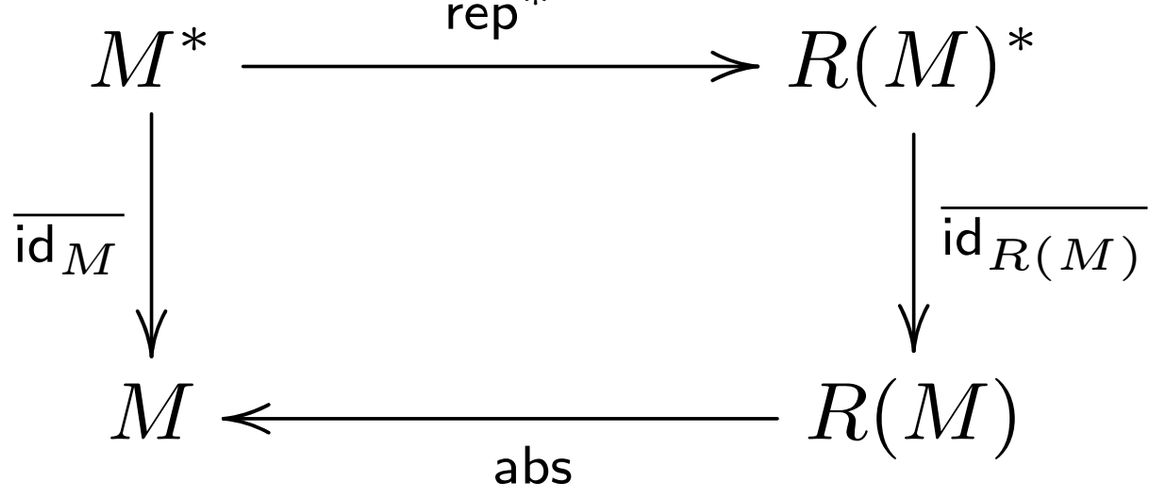
Cayley Representation

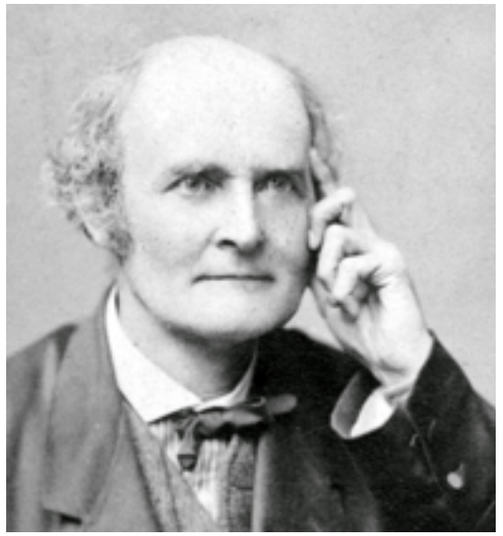
difference list
Hughes list

Original Program

```
rep(x) = λy.x ⊗ y
```

Better Representation





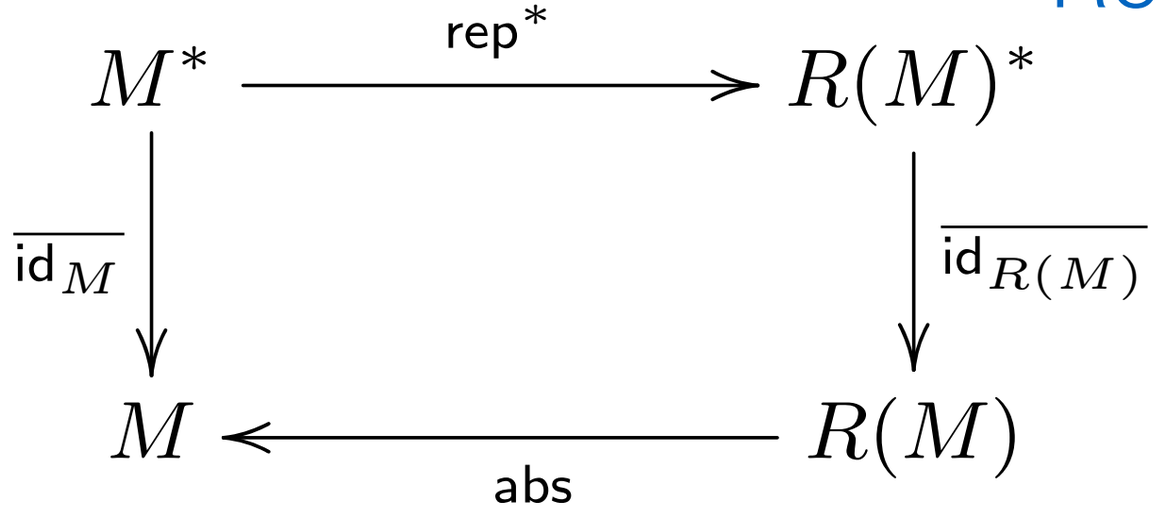
Cayley Representation

difference list
Hughes list

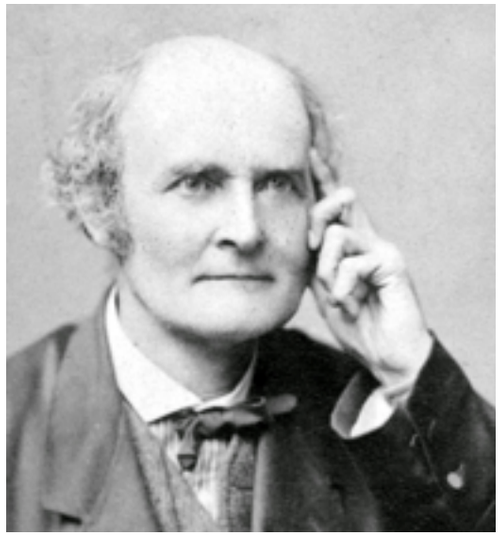
Original Program

```
rep(x) = λy.x ⊗ y
```

Better Representation




```
ε = id
⊗ = ∘
```



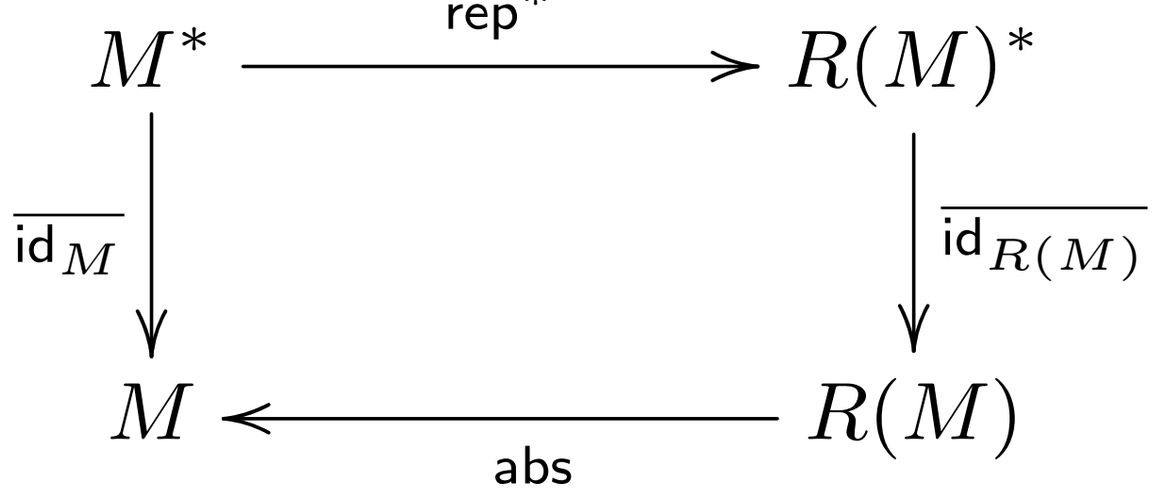
Cayley Representation

difference list
Hughes list

Original Program

```
rep(x) = λy.x ⊗ y
```

Better Representation




```
ε = id
⊗ = ∘
```

```
abs(rx) = rx ε
```

Near-Semirings

Near-Semiring
= 2 Related Monoids

Example

$(\mathbb{Z},*,1)$

$(\mathbb{Z},+,0)$

Near-Semiring
= 2 Related Monoids

Example

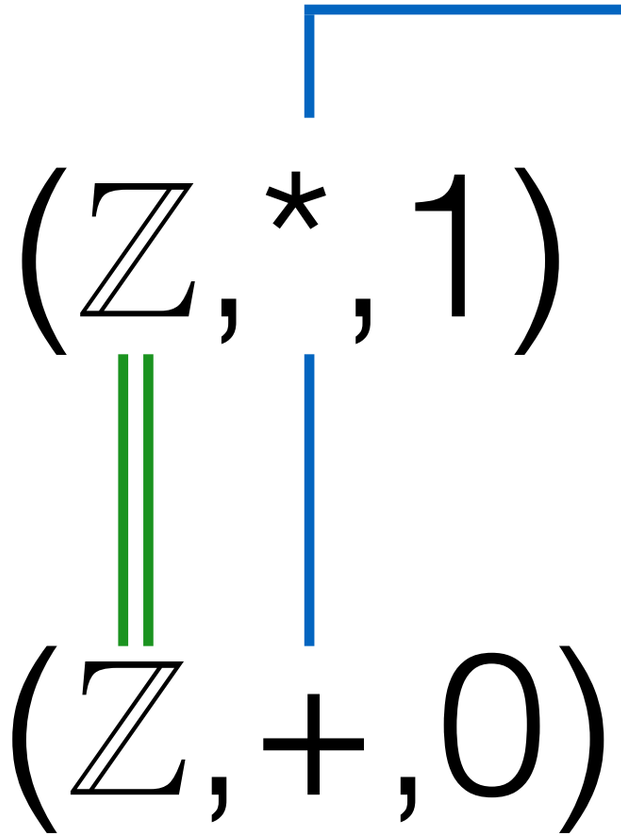
$(\mathbb{Z}, *, 1)$

$\|$

$(\mathbb{Z}, +, 0)$

Near-Semiring
= 2 Related Monoids

Example



(x + y) * z

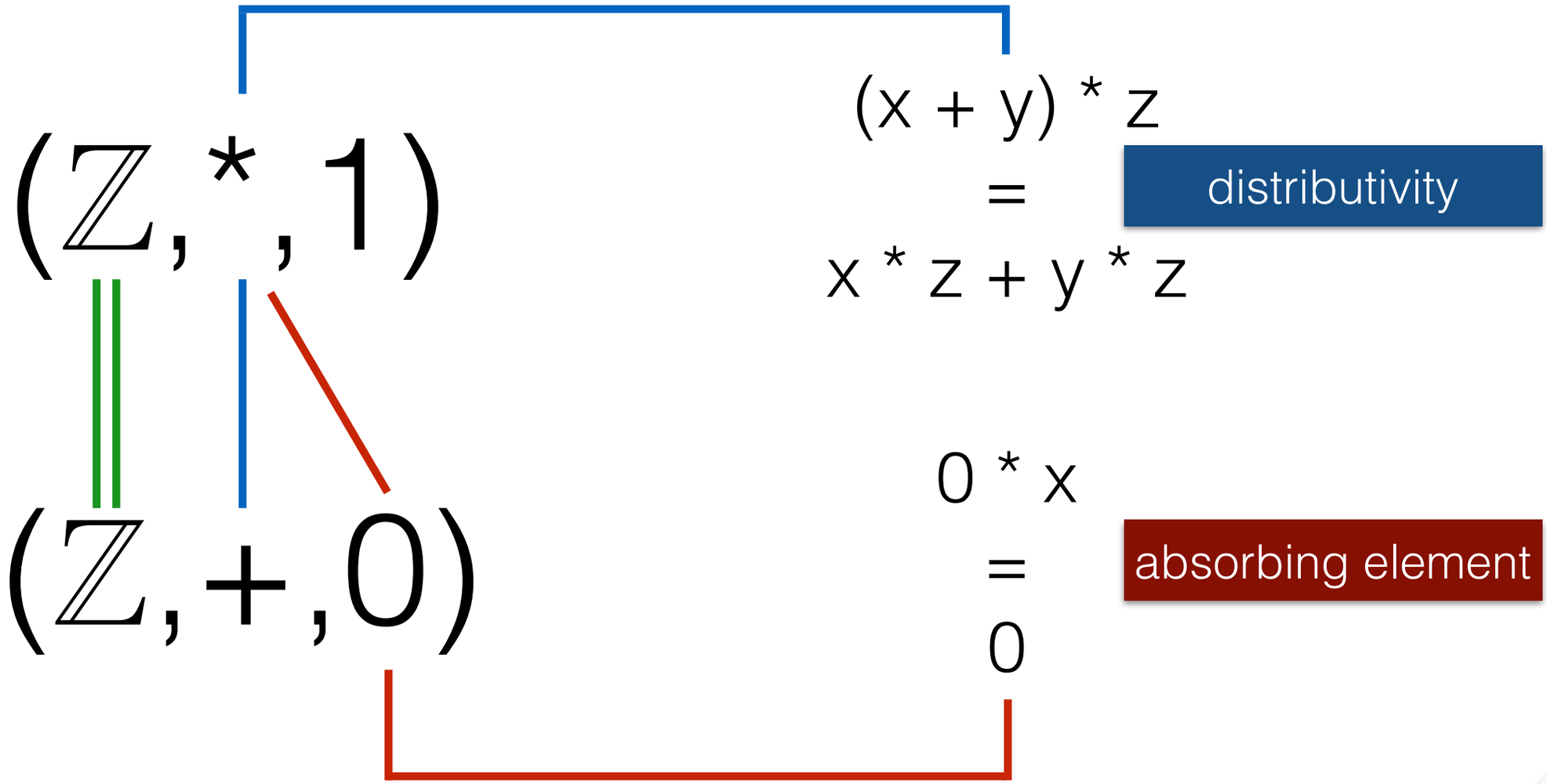
=

distributivity

x * z + y * z

Near-Semiring
= 2 Related Monoids

Example



Near-Semiring

$$(M, \otimes, \mathsf{e}, \oplus, \mathsf{z})$$

Near-Semiring

multiplicative structure

$$(M, \otimes, \mathsf{e}, \oplus, \mathsf{z})$$

Near-Semiring

multiplicative structure

$$(M, \otimes, \mathbf{e}, \oplus, \mathbf{z})$$

additive structure

Near-Semiring

multiplicative structure

$$(M, \otimes, \mathsf{e}, \oplus, \mathsf{z})$$

additive structure

2 x monoid laws

&

$$\mathsf{z} \otimes a = \mathsf{z}$$

$$(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$$

Near-Semiring

multiplicative structure

$$(M, \otimes, \mathsf{e}, \oplus, \mathsf{z})$$

additive structure

2 x monoid laws

&

$$\mathsf{z} \otimes a = \mathsf{z}$$

$$(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$$

$$\begin{aligned}
&\textbf{class}\ \text{Nearsemiring}\ a\ \textbf{where} \\
&\quad (\otimes) :: a \to a \to a \\
&\quad \text{one} :: a \\
&\quad (\oplus) :: a \to a \to a \\
&\quad \text{zero} :: a
\end{aligned}$$

Challenges

Free Near-Semiring?

Cayley Representation?

Near-Semiring Construction 1️⃣

Free near-semiring

```
data Forest a          data Tree a
  = Forest [Tree a]      = Leaf
                         | Node a (Forest a)
```

Near-Semiring Construction 1️⃣

Free near-semiring

```
data Forest a          data Tree a
  = Forest [Tree a]      = Leaf
                         | Node a (Forest a)
```

$$
\begin{aligned}
&\textbf{instance}\ \mathsf{Nearsemiring}\ (\mathsf{Forest}\ a)\ \textbf{where}\\
&\quad \mathsf{zero} = \mathsf{Forest}\ [\,]\\
&\quad \mathsf{one} = \mathsf{Forest}\ [\mathsf{Leaf}]\\
&\quad (\mathsf{Forest}\ xs) \oplus (\mathsf{Forest}\ ys) = \mathsf{Forest}\ (xs +\!\!+ ys)\\
&\quad (\mathsf{Forest}\ xs) \otimes (\mathsf{Forest}\ ys) = \mathsf{Forest}\ (\mathsf{concatMap}\ g\ xs)\\
&\qquad \textbf{where}\ g\ \mathsf{Leaf} = ys\\
&\qquad\quad g\ (\mathsf{Node}\ a\ n) =\\
&\qquad\qquad [\mathsf{Node}\ a\ (n \otimes (\mathsf{Forest}\ ys))]
\end{aligned}
$$

Near-Semiring Construction 1️⃣ ✅

Free near-semiring

```
data Forest a          data Tree a
  = Forest [Tree a]      = Leaf
                         | Node a (Forest a)
```

$$
\begin{aligned}
&\textbf{instance}\ \mathsf{Nearsemiring}\ (\mathsf{Forest}\ a)\ \textbf{where}\\
&\quad \mathsf{zero} &&= \mathsf{Forest}\ [\,]\\
&\quad \mathsf{one} &&= \mathsf{Forest}\ [\mathsf{Leaf}]\\
&\quad (\mathsf{Forest}\ xs) \oplus (\mathsf{Forest}\ ys) &&= \mathsf{Forest}\ (xs +\!\!+ ys)\\
&\quad (\mathsf{Forest}\ xs) \otimes (\mathsf{Forest}\ ys) &&= \mathsf{Forest}\ (\mathsf{concatMap}\ g\ xs)\\
&\qquad \textbf{where}\ g\ \mathsf{Leaf} &&= ys\\
&\qquad\qquad g\ (\mathsf{Node}\ a\ n) &&=\\
&\qquad\qquad\quad [\mathsf{Node}\ a\ (n \otimes (\mathsf{Forest}\ ys))]
\end{aligned}
$$

Near-Semiring Construction 2️⃣

The **double** Cayley representation

$$\textbf{type}\ \mathrm{DC}\ n = (n \to n) \to (n \to n)$$

$$\begin{aligned}
&\textbf{instance}\ \mathrm{Monoid}\ n \Rightarrow \mathrm{Nearsemiring}\ (\mathrm{DC}\ n)\ \textbf{where}\\
&\quad f \otimes g = f \circ g\\
&\quad \mathrm{one} \quad = \mathrm{id}\\
&\quad f \oplus g = \lambda h \to f\ h \circ g\ h\\
&\quad \mathrm{zero} \ = \mathrm{const\ id}
\end{aligned}$$

see paper for caveat

Near-Semiring Construction 2 ✓

The **double** Cayley representation

$$\textbf{type}\ \mathrm{DC}\ n = (n \to n) \to (n \to n)$$

$$\begin{aligned}
&\textbf{instance}\ \mathrm{Monoid}\ n \Rightarrow \mathrm{Nearsemiring}\ (\mathrm{DC}\ n)\ \textbf{where}\\
&\quad f \otimes g = f \circ g\\
&\quad \mathrm{one} \quad = \mathrm{id}\\
&\quad f \oplus g = \lambda h \to f\ h \circ g\ h\\
&\quad \mathrm{zero} \ = \mathrm{const\ id}
\end{aligned}$$

see paper for caveat

Generalised Monoids

Monoidal Category

Category **Set**		Category C
set		object
function		morphism
cartesian product		bifunctor $\otimes$
singleton set		object I

Monoidal Category

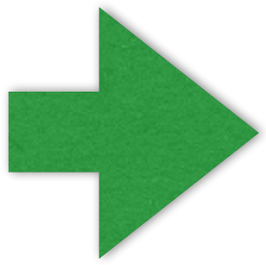
Category **Set**	Category C
set	object
function	morphism
cartesian product	bifunctor $\otimes$
singleton set	object I

monoidal category C

$$\alpha : A \otimes (B \otimes C) \cong (A \otimes B) \otimes C$$
$$\lambda : I \otimes A \cong A$$
$$\rho : A \otimes I \cong A$$

Generalised Monoid

Monoid
in **Set**



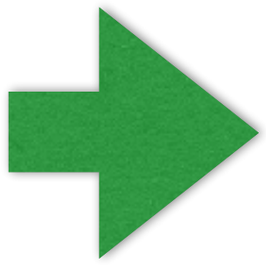
Monoid
in Category $\mathcal{C}$

(A, m, e)

(M, m, e)

Generalised Monoid

Monoid
in **Set**



Monoid
in Category $\mathcal{C}$

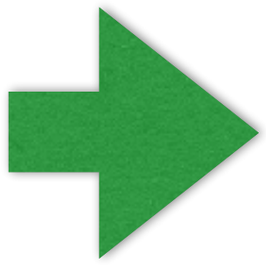
(A, m, e)

(M, m, e)

$$\lambda = m \circ (e \otimes \mathsf{id})$$
$$\rho = m \circ (\mathsf{id} \otimes e)$$
$$m \circ (m \otimes \mathsf{id}) \circ \alpha = m \circ (\mathsf{id} \otimes m)$$

Generalised Monoid

Monoid
in **Set**



Monoid
in Category $\mathcal{C}$

(A, m, e)

(M, m, e)

$$m : A \times A \to A$$

$$e : \{*\} \to A$$

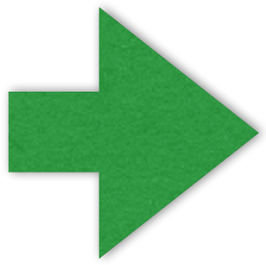
$$\lambda = m \circ (e \otimes \mathsf{id})$$

$$\rho = m \circ (\mathsf{id} \otimes e)$$

$$m \circ (m \otimes \mathsf{id}) \circ \alpha = m \circ (\mathsf{id} \otimes m)$$

Generalised Monoid

Monoid
in **Set**



Monoid
in Category $\mathcal{C}$

(A, m, e)

(M, m, e)

$$m : A \times A \to A$$

$$e : \{*\} \to A$$

$$m : M \otimes M \to M$$

$$e : I \to M$$

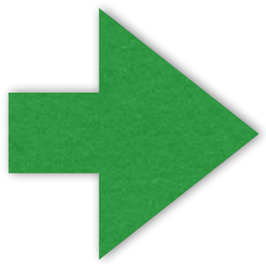
$$\lambda = m \circ (e \otimes \mathsf{id})$$

$$\rho = m \circ (\mathsf{id} \otimes e)$$

$$m \circ (m \otimes \mathsf{id}) \circ \alpha = m \circ (\mathsf{id} \otimes m)$$

Generalised Monoid

Monoid
in **Set**



Monoid
in Category $\mathcal{C}$

(A, m, e)

(M, m, e)

bifunctor

$$m : A \times A \to A$$

$$e : \{*\} \to A$$

$$m : M \otimes M \to M$$

$$e : I \to M$$

$$\lambda = m \circ (e \otimes \mathsf{id})$$

$$\rho = m \circ (\mathsf{id} \otimes e)$$

$$m \circ (m \otimes \mathsf{id}) \circ \alpha = m \circ (\mathsf{id} \otimes m)$$

Generalised Monoid

Monoid
in **Set**

(A, m, e)

$$m : A \times A \to A$$

$$e : \{*\} \to A$$

Monoid
in Category C

(M, m, e)

bifunctor

$$m : M \otimes M \to M$$

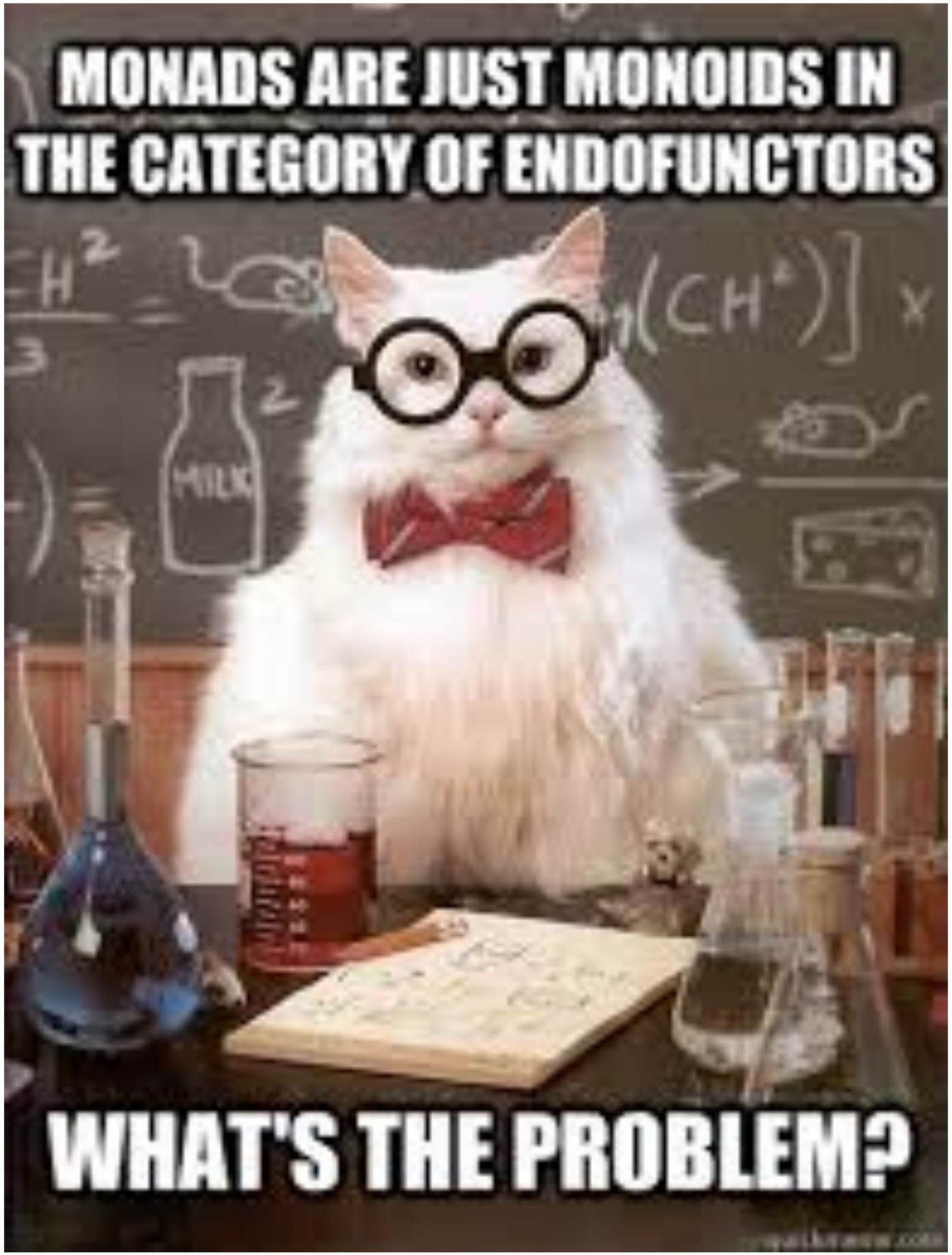
$$e : I \to M$$

object in C

$$\lambda = m \circ (e \otimes \mathsf{id})$$

$$\rho = m \circ (\mathsf{id} \otimes e)$$

$$m \circ (m \otimes \mathsf{id}) \circ \alpha = m \circ (\mathsf{id} \otimes m)$$

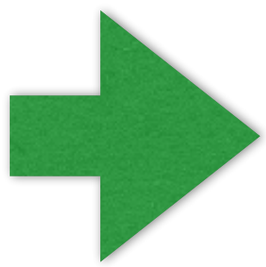

MONADS ARE JUST MONOIDS IN
THE CATEGORY OF ENDOFUNCTORS
MILK
WHAT'S THE PROBLEM?

Monoidal Category of Endofunctors

Set

$$m : A \times A \to A$$

$$e : \{*\} \to A$$



Endofunctors

$$m : M \circ M \dot{\to} M$$

$$e : \mathsf{Id} \dot{\to} M$$

Monoidal Category of Endofunctors

Set

$$m : A \times A \to A$$

$$e : \{*\} \to A$$

Endofunctors

$$m : M \circ M \dot{\to} M$$

$$e : \mathsf{Id} \dot{\to} M$$

Monoidal Category of Endofunctors

Set

$$m : A \times A \to A$$

$$e : \{*\} \to A$$

Endofunctors

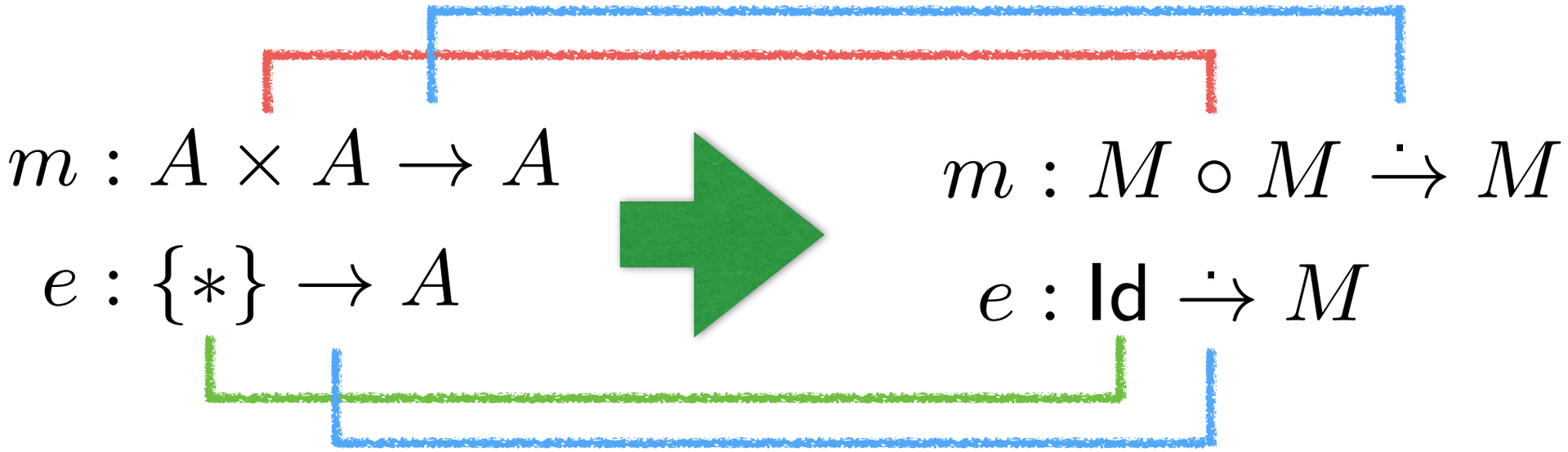
$$m : M \circ M \dot{\to} M$$

$$e : \mathsf{Id} \dot{\to} M$$

Monoidal Category of Endofunctors

Set

Endofunctors



From Theory to Haskell

$$m : M \circ M \dot{\to} M$$

$$e : \mathsf{Id} \dot{\to} M$$

From Theory to Haskell

$$m : M \circ M \dot{\to} M$$

$$e : \mathsf{Id} \dot{\to} M$$

```
join :: M (M a) -> M a
unit :: Id a -> M a
```

From Theory to Haskell

$$m : M \circ M \dot{\to} M$$

$$e : \mathsf{Id} \dot{\to} M$$

```
join :: M (M a) -> M a
unit :: Id a -> M a
```

```
class Monad m where
  (>>=) :: m a -> (a -> m b) -> m b
  return :: a -> m a
```

Monoidal Construction 1️⃣

Free monoid in the category of endofunctors

```
data Free f
  = Pure f
  | Op (f (Free f))
```

Monoidal Construction 1️⃣

Free monoid in the category of endofunctors

```
data Free f
  = Pure f
  | Op (f (Free f))
```

aka the free monad

Monoidal Construction 1️⃣

Free monoid in the category of endofunctors

```
data Free f
  = Pure f
  | Op (f (Free f))
```

aka the free monad

Algebraic Effect Handlers

★Handlers in Action, Kammar et al.
★Extensible Effects, Kiselyov et al.
★Fusion for Free, Wu & Schrijvers
★…

Monoidal Construction 2️⃣

The Cayley representation in the category of endofunctors

```
data CodT f a
  = CodT (∀ x. (a -> f x) -> f x)
```

Monoidal Construction 2️⃣

The Cayley representation in the category of endofunctors

```
data CodT f a
  = CodT (∀ x. (a -> f x) -> f x)
```

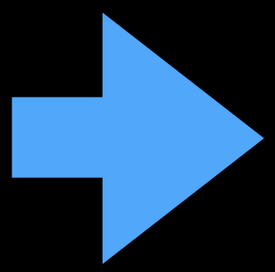
aka the codensity transformer

Monoidal Construction 2️⃣

The Cayley representation in the category of endofunctors

```
data CodT f a
  = CodT (∀ x. (a -> f x) -> f x)
```

aka the codensity transformer



★ Asymptotic improvement of computations over free monads, Voigtlaender

Generalized Near-Semirings

Near-Semiring in the Category of Endofunctors

Generic

$$m \quad : M \otimes M \to M$$
$$e \quad : I_{\otimes} \to M$$

$$a \quad : M \oplus M \to M$$
$$z \quad : I_{\oplus} \to M$$

Endofunctors

Near-Semiring in the Category of Endofunctors

Generic

$$m : M \otimes M \to M$$
$$e : I_{\otimes} \to M$$

$$a : M \oplus M \to M$$
$$z : I_{\oplus} \to M$$

Endofunctors

$$m : M \circ M \dot{\to} M$$
$$e : \mathsf{Id} \dot{\to} M$$

Near-Semiring in the Category of Endofunctors

Generic

$$m : M \otimes M \to M$$
$$e : I_{\otimes} \to M$$

$$a : M \oplus M \to M$$
$$z : I_{\oplus} \to M$$

Endofunctors

$$m : M \circ M \dot{\to} M$$
$$e : \mathsf{Id} \dot{\to} M$$

$$a : M \dot{\times} M \dot{\to} M$$
$$z : K_1 \dot{\to} M$$

Near-Semiring in the Category of Endofunctors

Generic

$$m : M \otimes M \to M$$
$$e : I_{\otimes} \to M$$

$$a : M \oplus M \to M$$
$$z : I_{\oplus} \to M$$

Endofunctors

$$m : M \circ M \dot{\to} M$$
$$e : \mathsf{Id} \dot{\to} M$$

lifted cartesian product

$$a : M \dot{\times} M \dot{\to} M$$
$$z : K_1 \dot{\to} M$$

Near-Semiring in the Category of Endofunctors

Generic

$$m : M \otimes M \to M$$
$$e : I_{\otimes} \to M$$

$$a : M \oplus M \to M$$
$$z : I_{\oplus} \to M$$

Endofunctors

$$m : M \circ M \dot{\to} M$$
$$e : \mathsf{Id} \dot{\to} M$$

$$a : M \dot{\times} M \dot{\to} M$$
$$z : K_1 \dot{\to} M$$

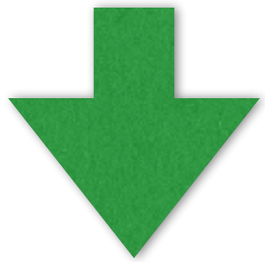
lifted cartesian product

constant functor

MonadPlus

$$a : M \dot{\times} M \dot{\to} M$$

$$z : K_1 \dot{\to} M$$

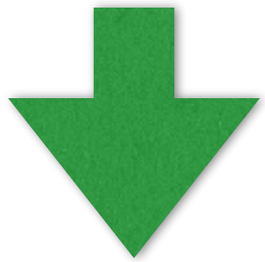


```
class Monad m => MonadPlus m where
  mplus  ::  m a -> m a -> m a
  mfail  ::  m a
```

MonadPlus

$$a : M \dot{\times} M \dot{\to} M$$

$$z : K_1 \dot{\to} M$$



```
class Monad m => MonadPlus m where
  mplus  ::  m a -> m a -> m a
  mfail  ::  m a
```

```
instance MonadPlus [] where …
instance MonadPlus Parser where …
```

Near-Semiring Construction 1️⃣

Free near-semiring in the category of endofunctors

```
data Forest f a
  = Forest [Tree f a]

data Tree f a
  = Leaf a
  | Fork (f (Forest f a))
```

Near-Semiring Construction 1️⃣✅

Free near-semiring in the category of endofunctors

```
data Forest f a
  = Forest [Tree f a]

data Tree f a
  = Leaf a
  | Fork (f (Forest f a))
```

Near-Semiring Construction 2️⃣

The double Cayley representation

$$\mathbf{newtype}\ \mathsf{DC}\ f\ x = \mathsf{DC}\ \{\mathsf{unDC} :: ((f \overset{\times}{\Rightarrow} f) \overset{\circ}{\Rightarrow} (f \overset{\times}{\Rightarrow} f))\ x\}$$

where

$$\mathbf{newtype}\ (\overset{\circ}{\Rightarrow})\ f\ g\ x = \mathsf{Ran}\ \{\mathsf{unRan} :: \forall y.(x \to f\ y) \to g\ y\}$$

$$\mathbf{newtype}\ (\overset{\times}{\Rightarrow})\ f\ g\ x = \mathsf{Exp}\ \{\mathsf{unExp} :: \forall y.(x \to y) \to (f\ y \to g\ y)\}$$

Near-Semiring Construction 2 ✓

The double Cayley representation

$$\mathbf{newtype}\ \mathsf{DC}\ f\ x = \mathsf{DC}\ \{\mathsf{unDC} :: ((f \overset{\times}{\Rightarrow} f) \overset{\circ}{\Rightarrow} (f \overset{\times}{\Rightarrow} f))\ x\}$$

where

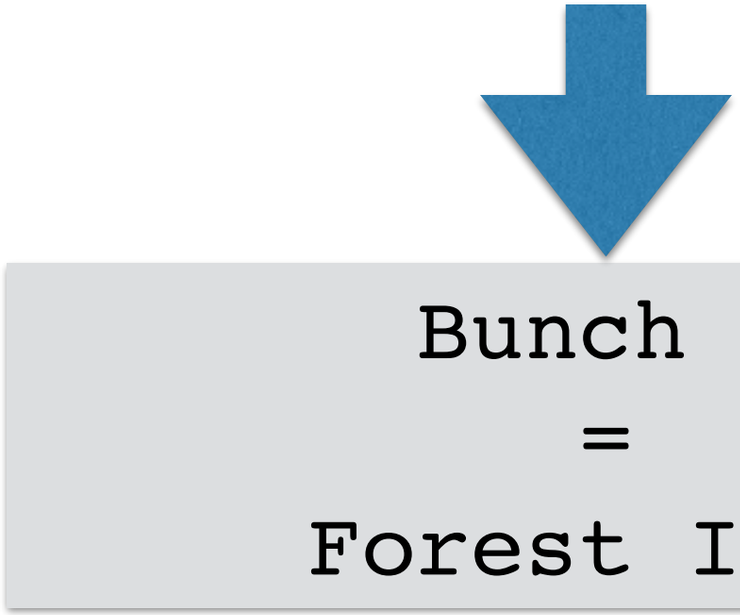
$$\mathbf{newtype}\ (\overset{\circ}{\Rightarrow})\ f\ g\ x = \mathsf{Ran}\ \{\mathsf{unRan} :: \forall y.(x \to f\ y) \to g\ y\}$$

$$\mathbf{newtype}\ (\overset{\times}{\Rightarrow})\ f\ g\ x = \mathsf{Exp}\ \{\mathsf{unExp} :: \forall y.(x \to y) \to (f\ y \to g\ y)\}$$

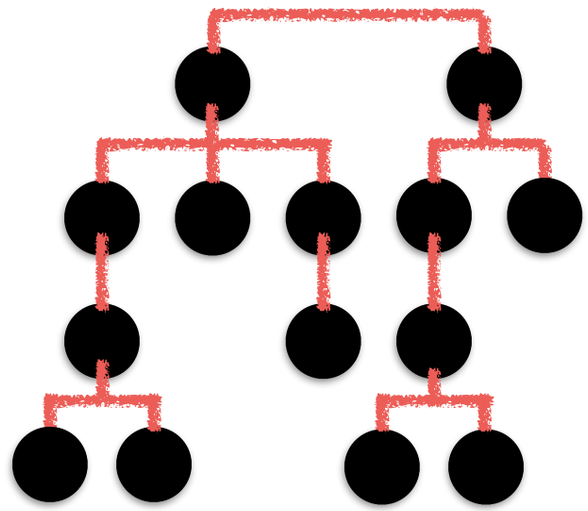
Application

Algebras for Combinatorial Search

Mike Spivey (JFP 2009)

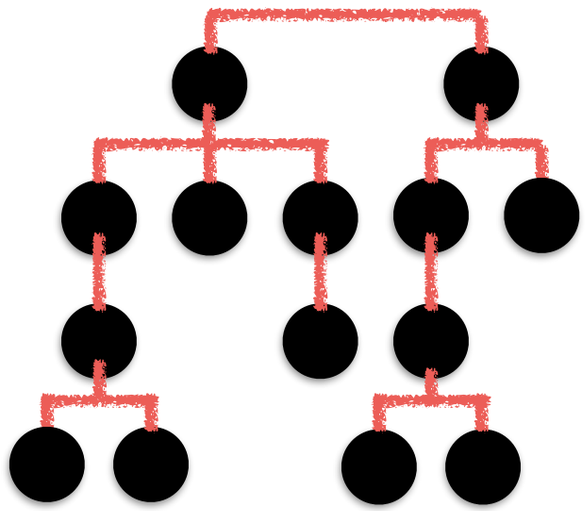


```
Bunch a
=
Forest Id a
```



Application

Algebras for Combinatorial Search
Mike Spivey (JFP 2009)



```
Bunch a
=
Forest Id a
```

Heuristics Entwined with Handlers Combined
Schrijvers et al. (PPDP 2014)

Benchmark

```
forest :: Int → Forest ()
forest 0 = return ()
forest n = (forest (n − 1) ≫= wrap′) `mplus` wrap′ ()
```

Benchmark

```
forest :: Int → Forest ()
forest 0 = return ()
forest n = (forest (n − 1) >>= wrap′) `mplus` wrap′ ()
```

Double left-nested recursion

Benchmark

```
forest :: Int → Forest ()
forest 0 = return ()
forest n = (forest (n − 1) >>= wrap′) `mplus` wrap′ ()
```

Double left-nested recursion

Size	Forest	DC Forest
50	2	0
100	26	1
250	569	8
500	5,472	36
1,000	60,070	172
2,500	T/O	1,484

Benchmark

```
forest :: Int → Forest ()
forest 0 = return ()
forest n = (forest (n − 1) >>= wrap′) `mplus` wrap′ ()
```

Double left-nested recursion

Size	Forest	DC Forest
50	$O(n^3)$ 2	0
100	26	1
250	569	8
500	5,472	36
1,000	60,070	172
2,500	T/O	1,484

Benchmark

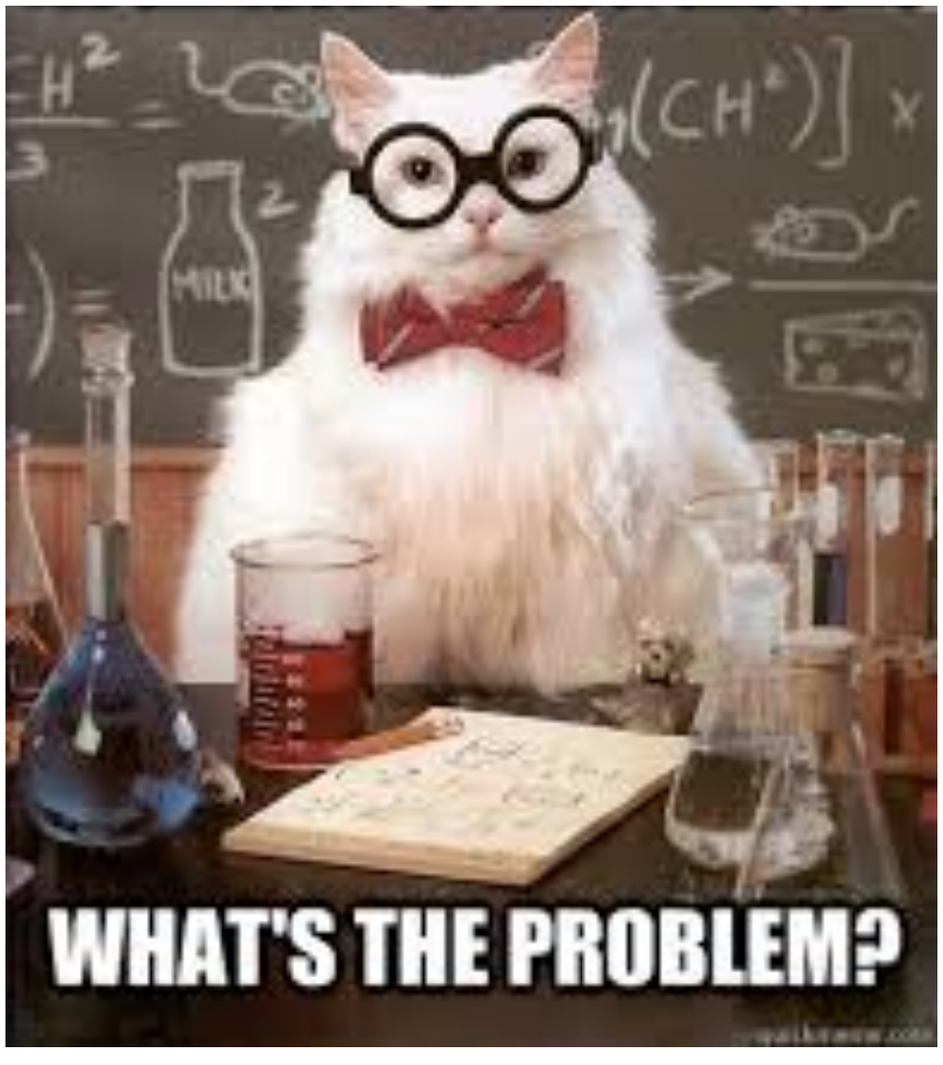
```
forest :: Int → Forest ()
forest 0 = return ()
forest n = (forest (n − 1) >>= wrap′) `mplus` wrap′ ()
```

Double left-nested recursion

Size	Forest	DC Forest
50	$O(n^3)$ 2	$O(n^2)$ 0
100	26	1
250	569	8
500	5,472	36
1,000	60,070	172
2,500	T/O	1,484

Summary

MonadPlus and Alternative
are just near-semirings
in the category of endofunctors.



In the Paper

- Generic Definitions
- Applicative Functors and Alternatives
- Application to interleaving parsers

Preprint

From monoids to near-semirings: the essence of MonadPlus **and** Alternative

Exequiel Rivas Mauro Jaskelioff
CIFASIS-CONICET
Universidad Nacional de Rosario, Argentina
jaskelioff@cifasis-conicet.gov.ar
rivas@cifasis-conicet.gov.ar

Tom Schrijvers
KU Leuven, Belgium
tom.schrijvers@cs.kuleuven.be

Abstract

It is well-known that monads are monoids in the category of endofunctors, and in fact so are applicative functors. Unfortunately, the benefits of this unified view are lost when the additional nondeterminism structure of MonadPlus or Alternative is required.

This article recovers the essence of these two type classes by extending monoids to near-semirings with both additive and multiplicative structure. This unified algebraic view enables us to generically define the free construction as well as a novel double Cayley representation that optimises both left-nested sums and left-nested products.

Keywords monoid, near-semiring, monad, monadplus, applicative functor, alternative, free construction, Cayley representation

1. Introduction

The monad interface provides a basic structure for computations:

```
class Monad m where
  return :: a → m a
  (>>=) :: m a → (a → m b) → m b
```

where the operations return, for injecting values, and >>=, for sequentially composing computations, are subject to the three monad laws. Study of this structure has led to many insights and applications. Two of these are especially notable:

1. The free instance of the monad interface, the free monad, has many applications in defining new monads and extending the capabilities of existing ones, e.g., in the form of algebraic effect handlers [16].

```
data Free f a = Return x | Op (f (Free f a))
instance Functor f ⇒ Monad (Free f) where
  return x  = Return x
  Return x >>= f = f x
  Op op     >>= f = Op (fmap (>>=f) op)
```

2. The codensity transformation CodT m provides a continuation-based representation for a monad m.

```
newtype CodT m a = CodT (∀x.(a → m x) → m x)
rep :: Monad m ⇒ m a → CodT m a
rep v = CodT (v>>=)
abs :: Monad m ⇒ CodT m a → m a
abs (CodT c) = c return
```

Since CodT m is a representation for m, we can lift m-computations to CodT m, compute in that monad, and when we are finished go back to m using abs. This change of representation is useful because it turns left-nested binds into right-nested binds [10, 22]. This is very convenient for monads (like the free monad) where left-nested binds are costly and right-nested binds are cheap.

```
instance Monad (CodT m) where
  return x      = CodT (λk → k x)
  CodT c >>= f = CodT (λk → c (λa →
                   let CodT g = f a in g k))
```

Both results can be derived by viewing a monad as a monoid in a monoidal category. The free monad is just the free monoid in that category and the codensity transformation arises as the *Cayley representation* of that monoid [17]. Moreover, useful generality is gained by this approach, as not only monads are monoids, but also applicative functors and arrows.

While the monad interface is well understood and comes with many useful results, it is also very limiting. When dealing with specific computations, we always require additional operations. A prominent example is non-determinism that occurs, e.g., in logic programming languages and parser combinators. Non-deterministic computations involve two additional operations: a failing computation (mzero) and a non-deterministic choice between two computations (mplus). These additional operations are captured in Haskell by the MonadPlus type class:

```
class Monad m ⇒ MonadPlus m where
  mzero :: m a
  mplus :: m a → m a → m a
```

This type class comes with five additional laws that govern the interaction between the operations.

It is not difficult to see that the above two constructions for the Monad interface do not work for the MonadPlus interface. Firstly, the free monad has no provision for the two additional operations of MonadPlus and the five additional laws.

Questions?